COMMUNITY DETECTION IN HYPERGRAPHS: EXPLORING MODULARITY MAXIMIZATION IN HIGHER-ORDER NETWORKS.

by

Abigail C. Headstrom

A Senior Honors Thesis

submitted to the faculty of Dartmouth College

in partial fulfillment of the requirements for the degree of

Bachelor of Arts

Department of Mathematics

Dartmouth College

June 2026

Abstract

Community detection partitions a network's nodes into tightly connected clusters, achieving many real-world applications. While these methods are well studied in pairwise graphs, many real-world systems involve higher-order interactions among multiple entities at once. Hypergraphs, which allow edges to connect more than two nodes, can capture such interactions. One approach to detecting communities in hypergraphs is the all-or-nothing (AON) model, which assumes an interaction is meaningful only if all of its nodes share a community. This thesis builds on the AON modularity framework of Chodrow et al. [1], extending it with partition-mapping techniques from the dyadic graph case. First, we address numerical instabilities in the parameter-estimation phase of that framework. Second, we introduce a partition-tracking mechanism and a method for re-checking previously discovered partitions to escape globally-suboptimal local optima. This in turn lets us define convergence criteria for the algorithm that previously relied on a fixed iteration count. Finally, we use the stored partitions to create a finite deterministic map to characterize the structure of the partition space. On the Enron email and Contact Primary School hypergraphs, the algorithms achieve diminishing discovery rates and the finite maps reveal consistent fixed points, even across runs that find non-overlapping partitions in th space. These results uncover a far richer partition space than a fixed iteration

budget can explore.

Preface

Acknowledgements

I would like to thank my advisor, Peter Mucha for his invaluable guidance, feedback, and support throughout this process and my undergraduate research journey. My thesis really started when I joined his research group my sophomore summer and I have learned so much under his guidance. I am so grateful for the Undergraduate Research at Dartmouth (URAD) program, which allowed me to experience mathematical research and explore my interest in network science. I do not think I would be writing a thesis without the experience I gained before my senior year. I would also like to thank my friends and my family. Their love and support means so much to me, and I could not have done it without them.

Contents

Chapter 1

Introduction

Networks are fundamental tools used to describe complex systems and their inter-network relationships. Comprised of nodes and their edges that connect them, these graphs describe complex dynamics with real-world applications. Networks can capture many types of systems and have been applied to problems such as neural networks [2], disease tracking [3], supply chain management [4], and social networks [5]. Understanding the underlying organizational structure is a common and important application of networks. Identifying groups of tightly connected nodes within these networks, referred to as communities, can provide important insights into the nature of a system. *Community detection* in networks is therefore a central component of network analysis.

The majority of community detection research has been developed for dyadic, or pairwise, networks. In this setting, a broad range of well-studied methods exist, including modularity maximization [6], spectral clustering [7], and generative probabilistic models such as the stochastic block model [8].

Many real-world applications, however, involve relationships that are not strictly

pairwise. That is, interactions within systems can involve more than two entities at the same time. For example, academic collaborations, group messaging threads, face-to-face contact patterns, and cellular signaling pathways all exhibit higher-order interactions that cannot necessarily be captured by pairwise edges alone. Representing these systems as ordinary graphs with only pairwise connections discards structural information that could be critical for identifying community structure within the networks.

One way to capture the nature of this relationship is through bipartite graphs, where one set of nodes represents a given relationship and the other set are the entities themselves. Consider, for example, a co-authorship network where one set of nodes represents the authors and the other denotes published papers. In this network, edges connect authors and the papers they collaborate on. Community detection in bipartite networks has been well studied, as shown in [9]. As Chodrow et al. [1] points out, however, bipartite representations of these interactions may fail to fully capture the nature of some higher-order relationships. The bipartite model uses the assumption that the membership of any two nodes in an edge are independent of one another. This may make sense in some networks; however, there are some cases where this assumption of independence undermines the underlying dynamics of the system. Consider a gossip network, which tracks the spread of confidential information [1]. Interactions in these networks require trust; the addition of one outlier in an interaction could completely halt the spread of information, making edge membership between nodes a dependent event.

Hypergraphs are an emerging framework in which hyperedges connect subsets of nodes rather than strictly pairwise interactions. They provide a structure for

representing higher-order interactions in a way that can honor complex, dependent relationships. Clustering in hypergraphs has applications in online social networks [10], circuit design [11], classifying psychiatric disorders [12], and semi-supervised learning [13].

One dynamic that hypergraphs are able to capture is the **all-or-nothing** (AON) system. This type of network, like the gossip network, assumes that an interaction can only occur if all nodes are nodes are present. In the context of community detection, it only considers edges in which all of its nodes belong to the same community. This thesis assumes AON dynamics in all of the graphs and preceding equations.

One avenue of research in hypergraph clustering has been through a combination of modularity maximization and generative clustering through the Stochastic Block Model [1]. This approach alternates between estimating possible community partitions and affinity parameters for the statistical model, following work in the dyadic graph case [14]. The work in [15] builds off of this method in the dyadic case, providing a natural starting place and theoretical grounding for extensions in the hypergraph case as well.

1.1 Contributions

This thesis offers an extension of the All-or-Nothing (AON) hypergraph modularity algorithm of Chodrow et al. [1], which uses alternating modularity maximization techniques. The core contributions of this work address (1) numerical correctness and stability of the modularity affinity parameters, (2) partition tracking and course-correction to escape local optima, and (3) finite deterministic partition mapping. These contributions build off of developed methods in regular graphs and apply them

to the hypergraph space.

The remainder of this thesis is organized as follows. Chapter 2 provides the necessary background on hypergraphs, graph-based community detection, and the landscape of hypergraph community detection methods. Chapter 3 describes the alternating modularity maximization approach in detail and grounds it within the current dyadic graph landscape. Chapter 4 presents the proposed modifications to the maximization algorithm and describes the partition tracking and check method. Chapter 5 discusses convergence and the added finite partition tracking phase. Chapter 6 reports the experimental results from the process and Chapter 7 concludes with a discussion of limitations and directions for future work.

1.2 Datasets

The work in this thesis uses two datasets: The Enron Email dataset, containing 148 nodes and 1,514 unique hyperedges [16] [17] and the Contact Primary School dataset, containing 242 nodes and 12,704 unique hyperedges [18] [19] [20]. The Enron set is derived from the emails of key employees at the Enron Corporation. It was selected for this work due to its potential for trust-based network dynamics and size of its edge and node sets. The Contact Primary School set tracks contact between primary school students and teachers. It was selected to mirror the work in [1].

Chapter 2

Background

This chapter provides the necessary mathematical background for the remainder of the thesis. We begin by introducing hypergraphs, provide a brief overview of community detection, and ground it in the modularity-based framework that motivates Chapter 3. Finally, we survey the broader, current landscape of community detection and situate the AON approach of Chodrow et al. [1] within the field.

2.1 Hypergraphs

Regular, or dyadic graphs, model pairwise relationships. In the simple graph case, this means each edge connects exactly two nodes. In the hypergraph case, edges can connect any number of nodes. Formally, a hypergraph $H = (V, E)$ consists of a set of nodes $V = \{1, \ldots, n\}$ and hyperedges $E \subseteq \mathscr{P}(V)$, where each hyperedge $e \in E$ is a subset of V with $|e| \geq 2$. We denote a k-edge as an edge of size exactly k. Note that a standard graph is just the case in which every hyperedge is a 2-edge. Throughout this thesis the minimum hyperedge size is 2. A hyperedge of size 1 would be a self-

loop, which are also commonly restricted for community detection in regular graphs as they do not provide information on inter-node relationships. Figure 2.1 illustrates a small hypergraph alongside its bipartite representation.

The **incidence matrix** $\mathbf{H} \in \{0,1\}^{n \times m}$ has $H_{ve} = 1$ if node $v \in e$ and 0 otherwise. The degree of node v is $\deg(v) = \sum_e H_{ve}$, interpreted as the number of hyperedges containing v.

Two alternative representations of a hypergraph frequently used in the community detection literature are bipartite [9] and clique expansion [21] representations. In a bipartite graph, a node v and a hyperedge-node e are connected by a graph edge if $v \in e$. The bipartite representation captures the full hyperedge structure but, as mentioned in Chapter 1 it assumes independence between nodes over membership in any given hyperedge. At the same time, bipartite graph algorithms are well studied for the hypergraph landscape [9].

The clique expansion method replaces each k-hyperedge with a k-clique of pairwise edges. This method captures the *total edge weight* between two nodes in a graph, but the edges (and their sizes) that the relationships come from become indistinguishable. At the same time, however, this abstraction allows any dyadic community detection algorithm to be applied immediately.

2.2 Community Detection

Community detection is the method of unsupervised clustering, which looks at ways to partition a network into groups of nodes that are more densely connected compared to the rest of the network. Formally, a partition $z : V \to \{1, \ldots, q\}$ assigns each node to one of q communities. The goal of community detection is to find a

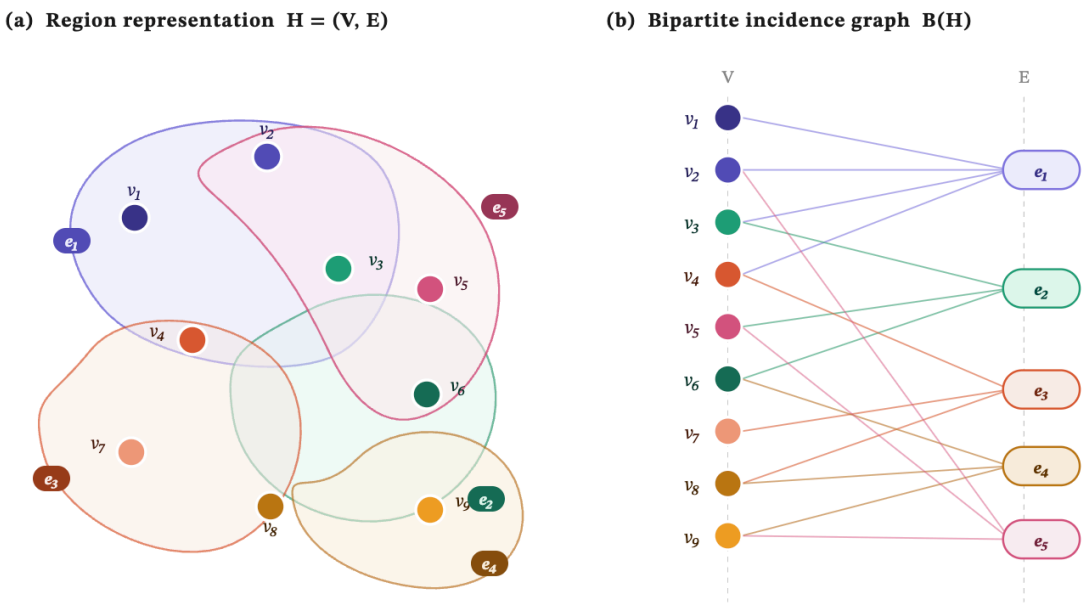


Figure 2.1: A small hypergraph (left) and its bipartite factor-graph representation (right). Each hyperedge node in the bipartite graph connects to each of the nodes it contains. Source: Claude Sonnet 4.6

partition that is optimal according to the specified model or function. In general, this problem is NP-hard [22], and algorithms often rely on heuristics, statistical inference, and spectral relaxations.

The field of community detection has been well studied in dyadic graphs. Girvan and Newman [23] introduced an algorithm based on inter-cluster connectedness of nodes and Newman [6] used the modularity function Q to motivate a spectral clustering algorithm, improving runtimes and clustering quality. The Stochastic Block Model (SBM) [8] uses statistical inference to generate graphs based on community structure and Newman [14] establishes a connection between modularity maximization and the SBM. The Louvain algorithm [24] provides a scalable greedy procedure for this modularity optimization that remains widely used. The stochasticity of the

Louvain algorithm leads to known degeneracies in modularity maximization [25] and other methods, leading to other improvements such as the Leiden algorithm [26]; however, the problem remains difficult.

2.3 Hypergraph Community Detection Landscape

The higher-order structure of hyperedges offers richer information about the system, but also introduces new computational challenges. Different-sized hyperedges can parse out the relative importance of the set of k-edges, or relationships of a given size, compared to the rest of the network. At the same time, the added edge sizes increases the complexity of the computations. Added parameter dimensionality rapidly increases computational complexity and removes flexibility within clustering algorithms. The extension of community detection to hypergraphs has grown significantly in recent years.

The simplest approach to hypergraph community detection is to project the hypergraph to a dyadic graph weighted by edge sizes and apply standard clustering algorithms [27]. While this method is able to leverage a more sophisticated toolbox of clustering methods, it discards information about hyperedge sizes, potentially losing important information about the network's dynamics [28]. Other approaches have looked at generative methods such as the planted partition model [29] and modularity-based methods [30]. This paper builds off of Chodrow et al. [1], which applies a maximization algorithm that combines parameter estimation from a modularity objective function and stochastic block model.

Chapter 3

AON Alternating Modularity Maximization

This chapter focuses on the All-or-Nothing alternating optimization pipeline for modularity in the Stochastic Block Model in [1], which is the foundation of the present work. This chapter starts by grounding the work in modularity maximization techniques in the dyadic graph case and demonstrates the Chodrow et al. [1] extension to hypergraphs. This extension uses the idea that pairwise dyadic graphs can be represented as a restricted case of hypergraphs where the maximum edge size is restricted to 2. An edge size of 1 would be a self-loop, which does not reveal anything about community structure in a graph. As such, modularity in hypergraphs applies the same principle to modularity equations from pairwise relations individually to each edge size in the graph.

The algorithm [1] uses an alternating maximization algorithm to find the best partition, z for the hypergraph. To do so, the All-or-Nothing optimization pipeline alternates between estimating two parameters: the Ω parameter, which is found using

maximum likelihood estimation and z, the partition. Hypergraph-adapted Louvain algorithms estimate the optimal partition z. The process alternates between these two estimations until the algorithm converges and the best partition is found.

3.1 Modularity Maximization and the Stochastic Block Model in Simple Graphs

Before developing the algorithm in hypergraphs, it is useful to ground its theory in the well-known dyadic graph case. The AON hypergraph pipeline generalizes the same alternating algorithm for graphs.

3.1.1 The Modularity Objective Function

Modularity is a quality function that measures how well a proposed partition of a network separates nodes into into connected communities. The intuition behind modularity is that partitions that place more edges *within* communities than what would be expected if placed at random over the same nodes are better or "more modular." The modularity function, referred to as Q, measures the excess within-group edges of a given partition relative to a null or random model. This is instead of simply counting the number of within-group edges, which would trivially be maximized by assigning all nodes to one community in a partition.

To define the function Q on an undirected, unweighted graph with n nodes and m edges, we first define the adjacency matrix as $\mathbf{A}$: $\mathbf{A}$ has entries $A_{ij} = 1$ if nodes i and j are connected and 0 otherwise. Given a partition explained by the group assignment vector $\mathbf{g}$, where g_i denotes the community label of node i, the modularity function is

given by:

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(g_i, g_j) \tag{3.1}$$

where $k_i = \sum_j A_{ij}$ is the degree of node i and $\delta(\cdot, \cdot)$ is the Kronecker delta function. The Kronecker delta ensures that only nodes placed in the same group are being counted. The null model term $k_i k_j / 2m$ represents the expected number of edges between nodes i and j under a random graph with the same expected degree sequence. Maximizing Q means finding the partition with the strongest within-cluster relationships *compared to the random baseline*. Under the modularity framework, this partition is considered the one with the best community structure.

This modularity equation, however, does not have any way to control for the number of clusters the function produces. Oftentimes networks can have multiple ways to cluster its nodes, depending on the preferred community size. In a friendship network, for example, community assignments to detect tight-knit friendships would require more, smaller communities compared to if we were interested in examining the large, broader divisions of a population. Both may have "correct" solutions, but require additional care in the modularity function.

The resolution parameter γ modifies the relative weight of the random-graph term. It controls the penalty for the number of clusters. A low γ value reduces the penalty, allowing larger communities to emerge; a higher γ value penalizes the null model more heavily, encouraging more small communities in the partition. The updated equation is as follows:

$$Q(\gamma) = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \gamma \frac{k_i k_j}{2m} \right) \delta(g_i, g_j) \tag{3.2}$$

At $\gamma = 1$, Equation 3.2 is the standard modularity function [14].

3.1.2 The Louvain Algorithm

The Louvain algorithm [24] is one of the most commonly used modularity partitioning algorithms. In a broad sense, it takes the modularity function $Q(\gamma)$ and the partition $\mathbf{g}$ and searches for a partition to produce the largest modularity score. It is a greedy heuristic, meaning it takes the most locally-optimal point at each step. It provides an efficient way to cluster a network, but can also get stuck in local optima easily.

The algorithm alternates between two distinct phases: the local moving phase and the contraction phase. In the first phase, the algorithm visits each node and computes the change in modularity resulting in a move from each one of its neighboring communities. The node is them moved (or stays in the same place) to the community with the highest change in modularity. In the second phase, every community collapses into one "supernode" and the process repeats iteratively. This is possible in regular graphs because edges can be turned into self-loops, but poses more of a challenge in hypergraphs, where there could be nodes within a hyperedge that span multiple communities. The Hypergraph Louvain algorithm in [1] addresses this issue and will be covered in a later section.

3.1.3 The Stochastic Block Model

Another popular used method used in graph clustering uses statistical inference and the Stochastic Block Model (SBM). Instead of maximizing a modularity function, this approach uses a generative model. It assumes the network was generated by a

probabilistic model with a community structure and then fits the model to the data. That is, the parameters driving the behavior of the model, including the partition itself, are fit to the model. Thus, the inferred parameters reveal the partition or community structure of the network.

In the SBM, n nodes are assigned to q groups, with g_i denoting the group membership of node i. The generative model "places" edges at random based on node group memberships. The affinity matrix Ω is then built, where ω_{rs} gives the expected number of edges between a node in group r and a node in group s. We then consider the following: given a partition $\mathbf{g}$ and affinity matrix Ω, the log-likelihood that the observed network (with adjacency matrix $\mathbf{A}$) was generated by the model is [14]

$$\log P(\mathbf{A} \mid \Omega, \mathbf{g}) = \frac{1}{2} \sum_{ij} \left(A_{ij} \log \omega_{g_i g_j} - \omega_{g_i g_j} \right) \tag{3.3}$$

Therefore, we find the best-fitting partition by maximizing Equation 3.3 over $\mathbf{g}$ and Ω.

An extension of the SBM is the degree-corrected stochastic block model (DCSBM). This model takes into account that node degrees do not always follow the same degree distribution, and it takes this into account. Newman [14] describes this change in detail and the final log-likelihood equation is given as

$$\log P(\mathbf{A} \mid \Omega, \mathbf{g}) = \frac{1}{2} \sum_{ij} \left(A_{ij} \log \omega_{g_i g_j} - \frac{k_i k_j}{2m} \cdot \omega_{g_i g_j} \right) \tag{3.4}$$

3.1.4 Alternating Modularity Maximization

The central discovery of Newman [14] is that the modularity equation is equivalent to the DCBM under certain restrictions. Specifically, this equivalence requires that

the affinity matrix Ω only take two values: ω_{in} for edges within a community and and ω_{out} for between-community edges. As Newman [14] describes, making this substitution allows the log-likelihood equation to simplify to an equation that is equivalent to the modularity function $Q(\gamma)$ up to a constant:

$$\gamma = \frac{\omega_{in} - \omega_{out}}{\log\omega_{in} - \log\omega_{out}} \tag{3.5}$$

Since we are interested in the parameters that maximize the two expressions, the constant is admissible. That is, maximizing modularity with the γ given by Equation 3.5 is equivalent to finding the maximum likelihood partition with the DCBM's special case. Additionally, this equivalence can tell us the correct value for γ. In practice, however, the values of ω_{in} and ω_{out} are unknown and must me estimated from the graph and existing partition $\mathbf{g}$. With two parameters to be estimated, γ and $\mathbf{g}$, this motivates using an alternating operation to estimate both. Given a current partition $\mathbf{g}$, we can use maximum likelihood estimation to compute ω_{in} and ω_{out} and the implied γ value. From there, the γ can be applied to the modularity function to re-estimate the partition $\mathbf{g}$ under the new parameters Newman [14]. This alternating optimization pipeline in regular graphs is the direct motivation for the hypergraph procedure in this work, which will be described in detail in the following section.

3.2 Alternating Modularity in Hypergraphs

In this section we demonstrate the extension of the regular case Chodrow et al. [1] implements in hypergraphs. Specifically, the methods use a degree-corrected hypergraph stochastic block model (DCHSBM) to derive a modularity function. Its

structure is a direct extension of the dyadic case, accounting for hyperedges of various sizes.

3.2.1 Modularity

Inserting the parameters under the restriction of the AON affinity function (see Chapter 1) into the general hypergraph modularity objective yields an the AON hypergraph modularity function [1]:

$$Q(z, \boldsymbol{\beta}, \boldsymbol{\gamma}) = -\sum_{k=2}^{\bar{k}} \beta_k \left[\mathrm{cut}_k(z) + \gamma_k \sum_{\ell=1}^{\bar{\ell}} \mathrm{vol}^{(\ell)k} \right] + J(\boldsymbol{\omega}) \tag{3.6}$$

where the cut term $\mathbf{cut}_k(z)$ counts the number of hyperedges of size k that span at least two different clusters. The volume term represents the null-model in a hypergraph and is explained in detail in [1].

The parameters β_k and γ_k are calculated from the AON affinity parameters ω_{k1} (within-cluster) and ω_{k0} (cross-cluster) as

$$\beta_k = \log\omega_{k1} - \log\omega_{k0}, \qquad \gamma_k = \beta_k^{-1}(\omega_{k1} - \omega_{k0}) \tag{3.7}$$

These parameters have direct applications to the parameter γ in the dyadic case. In Equation 3.6 β_k controls the weight assigned or relative importance of hyperedges of size k in the equation. For a given edge size k, γ_k is similar to γ in the regular case; it controls the cluster penalty for edges of that size. Excitingly, when a hypergraph is restricted to only edges of size $k = 2$, Equation 3.6 is exactly the standard dyadic modularity shown in Equation 3.2.

3.2.2 All-or-Nothing Louvain

The Louvain step for hypergraphs adds additional complexities compared to the dyadic case. Since edges can span more than two clusters, contracting to a supernode is not as easily computed in the hypergraph case. The hypergraph Louvain's second step still condenses clusters to supernodes, but it still must maintain the original hyperedge structure of all of the nodes in case one of them moves to another cluster in the next iteration. The AON simplification of the graph, however, makes this method possible. Since only hyperedges with all nodes in one cluster contribute positively to the objective function, the first step only has to count the number of edges that become fully internal or external with each reassignment move. This drastically improves the efficiency and makes the AON Louvain a feasible step in hypergraph modularity optimization. Further references to the Louvain algorithm refer to the AON hypergraph Louvain algorithm.

3.2.3 Hypergraph DCSBM

The Hypergraph DCSBM model (DCHSBM) assumes that the hyperedges placed on a node tuple R is a Poisson distribution. The details are outlined in Chodrow et al. [1]. Under the AON assumption, the affinity matrix Ω takes only two values per hyperedge size k:

$$\Omega(z_R) = \begin{cases} \omega_{k1} & \text{if all nodes in } R \text{ share the same label} \\ \omega_{k0} & \text{otherwise} \end{cases} \tag{3.8}$$

This mirrors the idea in the regular graph case from (Equation 3.5). The difference is that each edge size k has its own affinity pair $(\omega_{k1}, \omega_{k0})$. The affinity parameters

Algorithm 1 AON Alternating Optimization Pipeline [1]

Require: Hypergraph H, iteration budget N
Ensure: Best partition z_{best}
$z_0 \leftarrow$ initial partition via clique expansion Louvain
$\beta_1, \gamma_1 \leftarrow \beta(z_0), \gamma(z_0)$
for $i = 1, \ldots, N$ **do**
    $z_i \leftarrow \mathscr{L}(\beta_i, \gamma_i)$ ▷ AON Louvain step
    $q_i \leftarrow Q(z_i, \beta_i, \gamma_i)$ ▷ evaluate modularity
    $\beta_{i+1}, \gamma_{i+1} \leftarrow \beta(z_i), \gamma(z_i)$ ▷ update parameters
end for
return z_{best} ▷ best across all iterations

can be estimated from a given partition z via maximum likelihood estimation (MLE). From here, the β and γ parameters can also be updated.

3.2.4 Alternating AON Maximization

The alternating AON algorithm uses the modularity function and Louvain algorithm to repeatedly re-estimate the Ω and z parameters in an alternating manner. This method mirrors the work in Newman [14], where the affinity parameters Ω are updated from the current partition, and the derived parameters β_k and γ_k are recomputed via Equation 3.7. These parameters are then used by the Louvain step to recompute a new partition.

Chapter 4

Implementation and Extensions

The alternating All-or-Nothing maximization algorithm of Chodrow et al. [1] provides a strong foundation for hypergraph community detection, but several implementation challenges arise when applying it in practice. Because the Louvain algorithm is a greedy heuristic, it can return locally optimal partitions that are not globally optimal. These partitions can steer the rest of the algorithm towards suboptimal parts of the partition space. Work in dyadic graphs has addressed similar challenges by tracking partition histories [15]. This chapter proposes modifications to the original algorithm that build on these ideas. Additionally, it identifies and addresses numerical degeneracies within the current code from [31].

4.1 Tracking Partitions

In regular graphs, modularity degeneracy is a well-documented problem. The work in [25] shows that the modularity function Q can admit an exponential number of near-optimal, yet still sub-optimal, solutions. Oftentimes, this makes finding

a clear global maximum difficult to nearly impossible. Crucially, this degeneracy is not uniform across networks; the number of near-optimal partition grows with the strength of communities in the network. That is, networks with stronger community structures, specifically through the inter and outer-cluster edges, tend to produce more suboptimal, yet high-scoring partitions. Further, these high-modularity partitions can exhibit significant structural differences between them, producing significant challenges and consequences for picking a sub-optimal partition [25]. This presents a challenge for any algorithm that relies on small numbers of Louvain runs. In regular graphs, this exponential degeneracy occurs in only a couple of dimensions of parameters and one edge size. The hypergraph space occupies much higher dimensionality, which in theory could increase the potential degeneracies of a Louvain run.

In the hypergraph AON setting, the alternating optimization algorithm in [1] addresses this issue by running the Louvain step and parameter estimation repeatedly and selecting the partition, parameter pair with the highest likelihood across all iterations. The paper sets the fixed number of iterations at 20. While this approach pools solutions over the course of the run, it treats all iterations equally. That is, regardless of the quality and plausibility of the ith partition, the $i+1th$ parameters and subsequent partition are estimated from it. It omits the ability for cross-iteration feedback, which increases the potential for a sub-optimal partition to steer away from higher-quality regions of the space. Since $\beta_k = \log(\omega_{k1}) - \log(\omega_{k0})$ and γ_k both depend nonlinearly on the within- and cross-cluster affinity parameters ω_{k1} and ω_{k0}, a partition that misallocates even a small fraction of nodes can shift the estimated parameters and alter the partition found in the next iteration. Additionally, restricting the number of iterations too tightly could end the algorithm before it has been able

to thoroughly explore the partition space.

This section proposes three modifications to the original alternating optimization algorithm: (1) canonically relabeling each discovered partition to enable consistent cross-iteration comparison; (2) maintaining a set Σ of all unique partitions encountered; and (3) introducing a re-evaluation "check" step that, at each iteration, evaluates *all* unique partitions encountered against the current parameter estimates and starts the next step from the best scoring partition in Σ under the current parameters. Together, these modifications develop the algorithm from a single-path approach, to a spatial exploration that utilizes the storage of past partitions. This approach follows the framework of Gibson and Mucha [15], which shows that, in the dyadic graph case, maintaining a history of discovered partitions and re-evaluating each under the updated parameters can stabilize the iterative modularity maximization procedure.

4.1.1 Finding Unique Partitions

In order to understand convergence behavior of the maximization algorithm, it is important to understand how many *unique* partitions the Louvain step finds at each iteration. Accordingly, the first modification is to maintain a set Σ of unique partitions z such that, up to permutations of community labels, no $z_i = z_j$ for all $i \neq j$.

Comparing partitions from separate Louvain steps requires additional care beyond direct label comparisons. The Louvain algorithm assigns labels arbitrarily: two runs that produce structurally identical communities may assign them different integer labels, so a naive element-wise comparison would incorrectly classify them as distinct. To address this, we apply a canonical relabeling procedure to each newly discovered

partition before adding it to Σ. The procedure iterates through nodes in a fixed order, assigning community labels in the order they are first encountered. That is, the first node always receives label 1. Each subsequent nodes received the same labels as a previously seen node if it belongs to the same community, or the next unused label otherwise. This labeling is unique up to the node ordering, ensuring that structurally identical partitions can be identified in distinct iterations of the optimization procedure.

Following the relabeling step, the algorithm checks whether the newly returned partition $\hat{z}_i$ is already a member of Σ. If not, $\hat{z}_i$ is added to Σ regardless of if it is an optimal partition at the current parameters. A suboptimal partition at a given parameter pair could, potentially, be more optimal at a different set of parameters at a later iteration.

The rate of growth of $|\Sigma|$ over iterations can also be informative about the convergence of the algorithm. At the start of the runs, the growth rate, defined as $|\Sigma|/i$ where i is the number of iterations, is close to 1, as nearly all partitions discovered in the Louvain step are unique. As the iterations increase, the discovery rate slows as the algorithm discovers more repeat partitions. More observations about discovery rate are discussed in Chapter 6.

4.1.2 The "Check" Step

Tracking the unique partitions enables an important enhancement of the alternating loop. Rather than using the partition $\hat{z}_i$ returned by the current Louvain step to compute $(\beta_{i+1}, \gamma_{i+1})$, the algorithm re-evaluates all partitions in the set Σ. Using the current (β_i, γ_i) parameters allows a direct comparison of $\hat{z}_i$ with the rest of the

Algorithm 2 ReorderZ — Canonical Partition Relabeling

Require: Partition vector z of length n
Ensure: Canonically relabeled partition z_{new}

```
partition_map ← 0_|Σ|                          ▷ maps old label to new label
new_map ← 0_|Σ|
next_label ← 1
for each node v ∈ z do
    ℓ ← z[v]                                   ▷ get community label of node v
    if partition_map[ℓ] = 0 then               ▷ first time seeing label ℓ
        partition_map[ℓ] ← next_label
        next_label ← next_label + 1
    end if
    new_map[v] ← partition_map[ℓ]              ▷ assign canonical label
end for
return new_map
```

already-found partitions. If there is a partition that yields a higher modularity score than $\hat{z}_i$ with the current parameters, then $\hat{z}_i$ can not be the optimal partition at this point. Continuing to explore the path extending from this partition does not make sense if it is not optimal at its current parameters. Instead, the algorithm selects the globally optimal partition *within the current partition space*, adjusting the path of exploration back to the current optimal solution.

Concretely, after the Louvain step returns $\hat{z}_i$ and the corresponding parameters (β_i, γ_i) are computed, the check step evaluates $Q(\beta_i, \gamma_i, z_p)$ for every $z_p \in \Sigma$ and selects the globally optimal partition for the current parameters:

> Let $\hat{z}_i$ be the estimated partition returned by the Louvain algorithm under parameters β_i and γ_i. Then let $z_i \in \Sigma$ be the partition such that
>
> $$z_i = \underset{z_p \in \Sigma}{\arg\max}\, Q(\beta_i, \gamma_i, z_p). \tag{4.1}$$

The parameters for the next iteration, $(\beta_{i+1}, \gamma_{i+1})$, are estimated using z_i rather than $\hat{z}_i$. Note that $\hat{z}_i$ is still added to Σ before the check step executes.

The motivation for this adjustment is based on literature in the dyadic graph setting. Because the Louvain algorithm is a greedy heuristic, getting stuck in local optima that are not global optima is common [25]. If a suboptimal partition is used to re-estimate the parameters, they may push the next Loouvain step further from the globally optimal partition. At the same time, if a prior partition has a higher modularity score under the current parameters, it is better to use that partition for the next parameter update. In the dyadic graph setting, Gibson and Mucha [15] demonstrate a similar method, keeping only the "admissible" or "somewhere-optimal" partitions. The other ones are not considered. Even on small graphs, such as the Zachary Karate Club example with only a handful of nodes, Gibson and Mucha [15] found that less than 2% of all found partitions were anywhere admissible. In the hypergraphs, with higher-dimension parameter spaces, it is likely that the percentage of admissible partitions could be even lower.

4.2 Beta and Gamma Parameter Behavior

A key observation during implementation concerns the calculation of the γ_k parameters in `learn_omega_aon`. The original code in [31] computes γ_k as

$$\gamma_k = \log \omega_{k1} - \log \omega_{k0}, \tag{4.2}$$

which does not normalize each γ_k term by β_k as described in the γ formula in Chodrow et al. [1]. This calculation can be found on line 698 in the `learnOmegaAON` function in [31]. The correct equation is

$$\gamma_k = \frac{\log\omega_{k1} - \log\omega_{k0}}{\beta_k} \tag{4.3}$$

for each edge size k.

Without this normalization, the estimated γ_k values at each iteration are effectively meaningless with respect to the current partition. They still reflect the raw log-ratio of the affinity parameters, but their magnitude varies without any normalization from the β_k terms. As a result, each partition maps to a γ vector that is mismatched from the partition it came from. The next Louvain step then optimizes a partition from mismatched, or effectively random, parameters. This creates a feedback loop where neither the parameters nor partitions make directed progress in the estimation space, as they are bouncing around without normalization.

It is worth noting that the trajectories shown in Figure 4.1 are produced in the original alternating AON algorithm *without the check step*. That is, each parameter estimation follows from the most recent partition rather than the most optimal one in Σ. This reflects the baseline behavior of the algorithm as closely as possible and the raw trajectories of the parameters without the redirection in the check step.

4.2.1 Effect of the Uncorrected Parameters

Figure 4.1 shows the β_k and γ_k trajectories over 500 alternating iterations under the incorrect calculation of β_k. Both exhibit high-frequency noise across all edge sizes with no clear convergence or drift. While all of the trajectories stay within some mean range, their values do not settle and oscillate consistently and without any diminishing magnitude throughout all 500 iterations. This behavior is consistent with the feedback loop described above. Each γ_k value does not properly represent

the resolution penalty in the modularity function, so the Louvain algorithm at the next step is optimizing a completely mismatched set of parameters. This leads to a new parameter update from the Louvain's partition that has been driven into a completely disconnected space.

4.2.2 Effect of the Corrected Parameters

Adding the division by β_k at each estimation step restores the intended formula and re-establishes the connection between the partition and resolution parameters. Figure 4.2 shows the corresponding trajectories, still without the check step, so that the fix can be seen in isolation of the path-correction adjustments. There is a clear difference from Figure 4.1, where the noise in the parameters dampens and they start to follow clear, connected paths through the space. Some noise still occurs, but this is expected, given the Louvain's potential for finding suboptimal partitions. This also serves as further motivation for the "check" step proposed in Section 4.1.

4.3 Edge and Cut Degeneracies

A second implementation issue in the `learn_omega_aon` function concerns the numerical stability of the objective function $\mathbf{Q}$ around edge cases when the algorithm starts finding partitions with small numbers of clusters. During experimentation, there were two specific edge cases that cause the objective function to reach undefined values: (1) when the current partition places all edges of a given size k are either entirely within or entirely across clusters and (2) the Louvain selects a partition with one community, forcing the cross-cluster term to 0.

4.3.1 Loss of Cut or Uncut Edges

The algorithm estimats the affinity parameters ω_{k1} and ω_{k0} from the weighted counts of edges and cut edges of each size k. In the code from [31], these counts are accumulated in the matrix `EdgesAndCuts`, where the first row tracks total edge weight and the second row tracks the weight of cut edges.

If the weight of the cut edges for a given size k goes to 0, which means there are no edges of size k that are cut across clusters in a given partition, then $\omega_{k0} = 0$. This would them push $\beta_k = \log(\omega_{k1}) - \log(\omega_{k0})$ to ∞. The same issue happens if all of the edges of size k are cut, but $\omega_{k1} = 0$ instead. In both cases, γ_k becomes undefined and so does the modularity score. The conditions of the Louvain algorithm deny any node movement when the modularity score is undefined, which means the Louvain would repeatedly output a partition where all nodes are in their own community.

The original code in [31] already applies a small positive floor to `EdgesAndCuts`[1,:], which eliminates the degeneracy in ω_{k1}. In this thesis we also propose the change for `EdgesAndCuts`[2,:].

4.3.2 Collapse to One Cluster

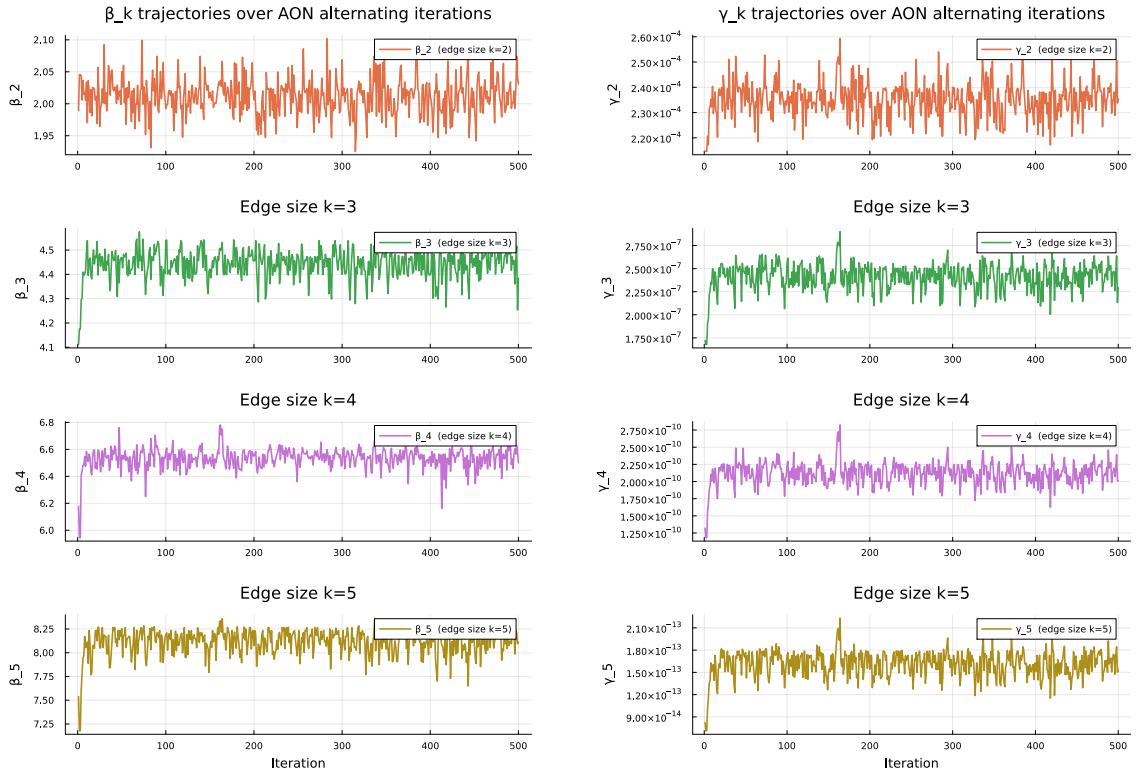
Once the fix allows the algorithm to reliably handle partitions with two clusters, the alternating optimization can spend more time exploring a two-cluster region of the space. In some cases, this can lead to a second degeneracy, where all nodes are placed in one cluster. In this case, since there are no other clusters in which the edges can form cuts which makes the β and γ vectors undefined. To address this issue, the proposed algorithm includes a step to check if a given partition has exactly one cluster. If so, it skips the parameter estimation for that iteration, automatically

making $\beta_{i+1} = \beta_i$ and $\gamma_{i+1} = \gamma_i$. This methodology is consistent with [32], which skips the parameter re-estimation step when the number of clusters reduces to 1 in the code application of the CHAMP algorithm.

4.4 Random Node Evaluation

The Louvain step from Chodrow et al. [1] offers the option to evaluate nodes in the same order every time or randomize the order in which the algorithm sees the nodes. Sine the Louvain is a greedy heuristic, the order in which it reads the nodes can impact the outcome. In the dyadic graph case, Motschnig et al. [33] found that some fixed node orderings led to worse modularity outcomes in the Louvain algorithm, suggesting a fixed ordering could encourage similar Louvain results across runs. One way to mitigate this outcome is to change the ordering at each step. The original code from Chodrow et al. [1] fixes the node order for all iterations; the alternating iterations often stop finding new partitions in the first handful of runs. In this paper, the proposed algorithm feeds the Louvain a randomized node order at each step to avoid the possibility of convergence caused by node orderings.

4.4 Random Node Evaluation

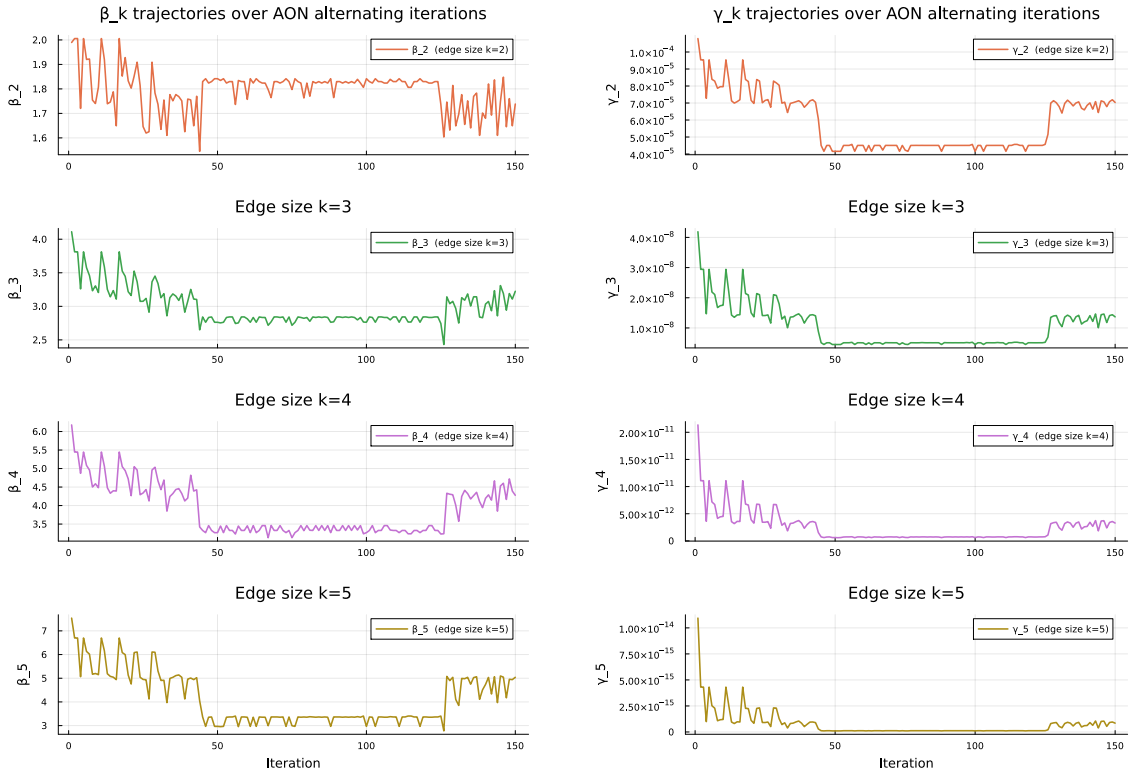


(a) β_k trajectories over 500 AON alternating iterations under the uncorrected γ_k calculation. All four edge sizes exhibit persistent, noisy oscillation with no convergence.

(b) γ_k trajectories over the same run. The similar noise pattern confirms that the miscalculated γ_k propagates instability back into subsequent parameter estimates at each iteration.

Figure 4.1: **Enron dataset:** Parameter trajectories under the uncorrected γ_k values run *without* the check step. Neither β_k nor γ_k converge over 500 iterations, consistent with the algorithm cycling through mismatched partition-parameter pairs rather than progressing toward a fixed point.

4.4 Random Node Evaluation



(a) β_k trajectories over 150 AON alternating iterations under the corrected γ_k calculation. All four edge sizes show more stable behavior, with parameters tracking the partition space rather than cycling arbitrarily.

(b) γ_k trajectories under the corrected formulation. Residual oscillation reflects the stochasticity of the Louvain step in the absence of the check step.

Figure 4.2: **Enron dataset:** Parameter trajectories under the corrected γ_k formulation, again run without the check step. Parameters now track the partition space coherently. Remaining oscillation is attributable to the stochasticity of the Louvain heuristic rather than to parameter miscalculation.

Chapter 5

Convergence and Partition Tracking

5.1 Partition Discovery Rate and Convergence

In Chodrow et al. [1], convergence does not have a concrete definition. They write, "We alternate between these two stages [parameter and partition estimation] until convergence," but do not explicitly define what convergence entails. There are multiple possibilities for what a converging alternating algorithm could mean. Since there are two distinct values to estimate - the β and γ arrays and the partitions themselves - either could serve as the basis for a stopping criterion.

One natural measure of convergence is the production of new partitions. If the Louvain step stops producing new partitions, it could indicate the current partition space has been fully explored. This would not necessarily mean all viable partitions have been found, but that the fixed point(s) may have reached a stable state where all of the near-optimal partitions reachable from them have been identified. A second possibility is the stability of the β and γ parameters. If these parameters stop moving, it could mean the algorithm has settled on an optimal β and γ space and

it has converged. Under the "check-step" algorithm, this condition corresponds to a prolonged time where the algorithm is picking a single partition in Σ. We call this phenomena "dwell length."

Both criteria share a fundamental limitation, however. Because the Louvain step is a greedy heuristic, we cannot guarantee that it will not find a new, higher-modularity partition in a later iteration. Parameter and partition stability at a given iteration are not enough to rule out future improvement. Additionally, reaching a state in which no new partitions are found may be highly unlikely. As discussed in Section 4.1 networks with more modular structures tend to produce exponentially more near-optimal solutions, many of which are similar and close to dominant partitions Good [25]. As a consequence, the Louvain step may continue to discover partitions with just slight differences from the dominant ones beyond when the algorithm has "converged" or found the optimal partitions. Additionally, in the dyadic graph case, [15] finds that the majority of *admissible* partitions are found early on in the Louvain iterations. From the Zachary Karate Club dataset, Gibson and Mucha [15] finds "the number of partitions in CHAMP's admissible subset and the main features of the corresponding domains of optimality typically stabilize rapidly in practice, with only slow or modest improvement upon adding further candidate partitions." They found that after around 100 iterations of the Louvain on the graph, the partition domains were all qualitatively similar. Note that the Karate Club dataset is a relatively small graph, so the number of iterations to find meaningful partitions can not necessarily compare. Rather, it indicates that, in the context of a graph's size, more meaningful partitions may be found earlier on in the iterations.

A third measure of convergence is the discovery rate of new partitions over time.

The running or cumulative discovery rate tracks the rate at which new unique partitions are found over time, aiming to capture the time at which the algorithm slows its learning rate to partitions that are, presumably, insignificant. If the algorithm has largely exhausted the accessible portion of the partition space, the discovery rate will fall towards zero and, for the reasons mentioned above, the Louvain calls are unlikely to materially produce candidates that change the finite map on Σ. The example in Gibson and Mucha [15], uses a graph that is considerably smaller than any graphs in this paper - with only 34 nodes - and the Louvain algorithm only considers pairwise relationships between nodes. In larger, higher-order graphs we can expect the algorithm to take significantly longer than 100 iterations to settle on the meaningful partitions, but this cutoff decision still follows the logic and intuition developed in Gibson and Mucha [15].

Once the cutoff is reached, the "exploration" part of the clustering methods are done. The intention of finding a convincing point of convergence is to obtain a finite deterministic set of partitions to describe the modularity space of the hypergraph. Once this discrete map has been found, we can analyze its behavior and stability of the partitions it contains.

Algorithm 3 demonstrates the whole exploration phase algorithm, including partition and discovery rate tracking.

5.2 Finite Deterministic Partition Mapping

The second phase of the proposed algorithm is the finite deterministic mapping or *restriction* phase. At this point, the alternating maximization algorithm in the

Algorithm 3 AON "Check-Step" Exploration Algorithm

Require: Hypergraph H, discovery rate threshold τ
Ensure: Set of unique discovered partitions Σ

$\Sigma \leftarrow \{z_0\}$, $i \leftarrow 1$ ▷ $\Sigma_i = \{z_1, \ldots, z_n\}$ is the set of unique partitions
$z_0 \leftarrow$ EstimateInitialPartition(H)
$\gamma_1, \beta_1 \leftarrow \beta(z_0), \gamma(z_0)$ ▷ initialize parameters from starting partition
while discovery rate $> \tau$ **do**
  $\hat{z}_i \leftarrow$ Louvain(β_i, γ_i) ▷ run Louvain-style search under current parameters
  $\Sigma \leftarrow \Sigma \cup \{\hat{z}_i\}$ if $\hat{z}_i \notin \Sigma$ ▷ add $\hat{z}_i$ only if it is a new unique partition
  $z_i \leftarrow \arg\max_{z_p \in \Sigma} Q(\beta_i, \gamma_i, z_p)$ ▷ select best partition across all discovered so far
  $\beta_{i+1}, \gamma_{i+1} \leftarrow \beta(z_i), \gamma(z_i)$ ▷ update parameters via MLE from winning partition
  discovery rate $\leftarrow |\Sigma| / i$
  $i \leftarrow i + 1$
end while
return Σ

"exploration" phase has outputted a finite set, Σ, which has converged to a complete set of unique partitions. Because Σ is finite, it is possible to evaluate the optimal β and γ parameters for each partition and its corresponding modularity score. This transforms the stochastic nature of the alternating problem into a deterministic, or non-random, finite map $\sigma : \Sigma \to \Sigma$, as shown in Algorithm 4. For any partition $z_p \in \Sigma$, we define the map $\sigma(z_p) = \arg\max_{z \in \Sigma} Q(z, \beta(z_p), \gamma(z_p))$. Every partition in Σ follows a trajectory under σ, mapping to another state — that is, another partition — in the set which maps to another one and so on. Note that a trajectory mapping a state to itself is still a viable event in the map.

The restriction phase operates in a similar manner to the estimation phase, except that the Louvain partition discovery step is removed. With it, all stochasticity is eliminated from the process. The parameter estimation and check phases are both deterministic. That is, for any partition $z_p \in \Sigma$, the parameters $\beta(z_p)$ and $\gamma(z_p)$ have a single possible solution determined by the maximum likelihood equation for the problem. Additionally, the check phase $\sigma(z_p)$ is evaluated by a simple check over the

whole set. Clearly, the domain of σ is finite because Σ is a finite set. The image of σ is also contained by this discrete set, so it is also finite. As such, every trajectory or path under σ must eventually revisit a state. This guarantees three possible types of trajectories in the map: fixed points, cycles, and transient paths leading into them. A fixed point is any partition z_p where $\sigma(z_p) = z_p$; cycle is a partition that does not satisfy the prior condition, but its trajectory eventually maps to itself; and a fixed point.

Each classification provides insights into the quality of the partition in the map. A transient partition with no in-degree is a partition that is nowhere optimal in the set of parameter values obtained from the other partitions. Even with its maximized parameters, there is at least one other partition with a higher modularity score under the same parameters. Transient partitions with an in-degree of at least one may be optimal compared to another partition, but they still lead to a point in the space that is more optimal. We call any point with in-degree at least one a "notable" point. Cycles in the map indicate more interesting behavior. In this case, every partition maps to another more-optimal partition, but the trajectory eventually returns to each partition. A fixed point describes a partition for which there is no other element in the set that has a more-optimal modularity score. It is also a "stable" point because its trajectory does not go off to any other state.

5.2.1 Partition Mapping in Pairwise Graphs

Evaluating the finite deterministic map has been well-studied in the pairwise graph setting. Since the modularity function $Q(z, \gamma)$ is linear (see Chapter 3), the modularity of each partition can be drawn as a straight line in the one-dimensional γ

parameter space. Therefore, the partition achieving the highest modularity at any given γ is determined by the upper envelope of the collection of lines.

The Convex Hull of Admissible Partitions (CHAMP) algorithm developed by Gibson and Mucha [15] exploits this geometry directly. Given an input set Σ of partitions, CHAMP computes the upper envelope of the corresponding linear modularity functions. Each partition z is "admissible" or "somewhere optimal" if and only if its line lies on the upper envelope of graph. Partitions whose lines fall below the envelope everywhere are nowhere-optimal and are discarded. This pruning step helps define the important parts of the partition space. In many applications, the majority of partitions in Σ are nowhere optimal and eliminated by CHAMP. This leaves a small subset of partitions that highlights the important partitions and potential solutions.

5.2.2 Hypergraph Partition Mapping

The higher dimensionality in the hypergraph space compared to the dyadic case poses significant computational challenges. Even when we restrict the modularity equation to edges of maximum size 5, there are 8 dimensions of the parameter space to optimize over — coming from the two vectors of length 4. In this high-dimension space, the curse of dimensionality quickly poses significant computational restrains on the optimization problem. Instead, we construct the finite deterministic map by exhaustive evaluation over Σ. The map is still able to identify fixed points, cycles, and transient points, which all provide meaningful insights into the quality of each partition. Understanding the domain of optimality requires more investigation, as these nonlinear high-dimensional parameter spaces pose more complex computational problems. In the case where we restrict to $2 \leq k \leq 5$, we still have an 8-dimensional

Algorithm 4 Deterministic Discrete Partition Mapping — "Restriction Phase"

Require: Set of discovered partitions Σ
Ensure: Fixed points, cycles, and never-returned-to points of the map σ

Let σ be the discrete, well-defined map $\sigma : \Sigma \to \Sigma$
for each partition $z_p \in \Sigma$ **do** ▷ repeat for all discovered partitions
    $\beta_{p+1}, \gamma_{p+1} \leftarrow \beta(z_p), \gamma(z_p)$ ▷ compute best parameters from z_p via E-step
    $z_{p+1} \leftarrow \arg\max_{z \in \Sigma} Q(z, \beta_{p+1}, \gamma_{p+1})$ ▷ find maximum-modularity partition under these parameters
    $\sigma(z_p) := z_{p+1}$ ▷ define the mapping from z_p to its image
end for
Classify each $z_i \in \Sigma$:
    Fixed point: z_i is a fixed point if $\sigma(z_i) = z_i$
    Cycle: z_i belongs to a cycle if $\sigma^k(z_i) = z_i$ for some $k > 1$
    Transient: no $z_j \in \Sigma$ satisfies $\sigma(z_j) = z_i$

parameter space.

Chapter 6

Results

6.1 Parameter Behavior in the Exploration Phase

Following the check-step implementation, the β and γ behavior evens out significantly. Figure 6.1 shows the first 500 iterations of the algorithm. Here, they jump between multiple different values, but they stay at the same values for prolonged amounts of time. This is because the algorithm is spending time returning to the same partition over and over, which keeps the parameter trajectories stable. This image demonstrates significantly more stability compared to the parameter trajectories prior to implementing this course-correcting check step.

The cluster counts and normalized mutual information (NMI) over time (Figures 6.2 and 6.3) provide a picture of the Louvain algorithm's behavior. While both datasets are shown on different timescales, they provide insights into the path at which the algorithm takes through the space; the Louvain algorithm is not often finding the same partition, or even number of communities, over and over. Both of these datasets have two clear cluster counts they consistently return to: 4 and 5 for

(a) β trajectories

(b) γ trajectories

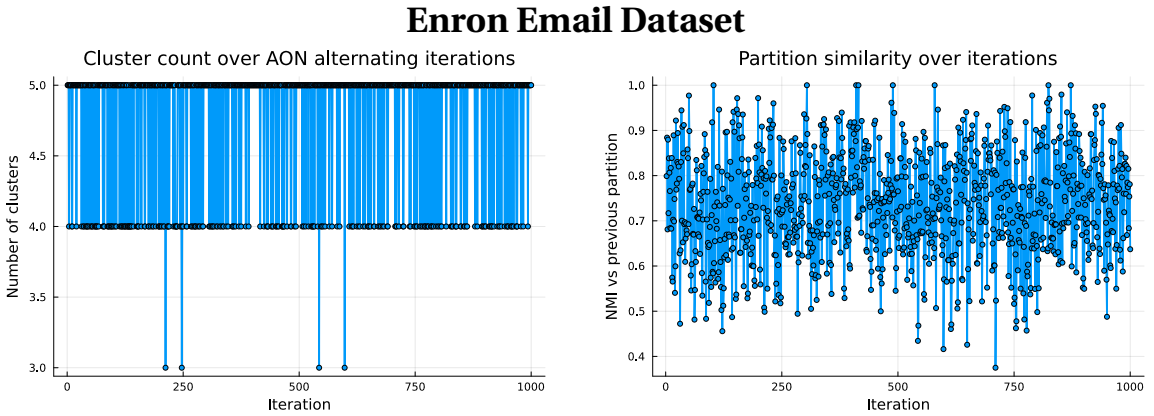
Figure 6.1: Trajectories of the learned parameters β (log-odds of within- vs. cross-cluster affinity) and γ (scaled cluster penalty) across alternating optimization iterations, shown per hyperedge size k. Both parameter sets move between stable states on intervals that increase overtime.

the Enron dataset and 3 and 4 for Primary School. Notably, in both of these cases the algorithm oscillates between the two options rather than staying at one number for a while and moving back. This also provides a strong argument for the value of the course-correcting methods behind the check step. Since the Louvain's algorithm is finding so many new partitions, even under optimal affinity parameters, the algorithm could easily direct itself into a suboptimal space.

The mutual information graphs provide an idea of how consistent the Louvain algorithm is when finding partitions. in the Enron dataset, there are very few instances where the NMI score is 1, which would indicate the Louvain found an identical partition to the one before. This indicates that the Louvain is finding many close-to-optimal points without actually re-finding the optimal one. Consistent with the the Primary School graph shows this as a much more consistent phenomena, but still shows a wide array of values. These graphs provide evidence for the importance of tracking all partitions over many runs. That is, since the Louvain algorithm is returning so many diverse partitions, even under the same affinity parameters, it needs time to find as many potential partitions as possible and an eventual mapping stage to make sense of them.

6.2 Convergence Measures

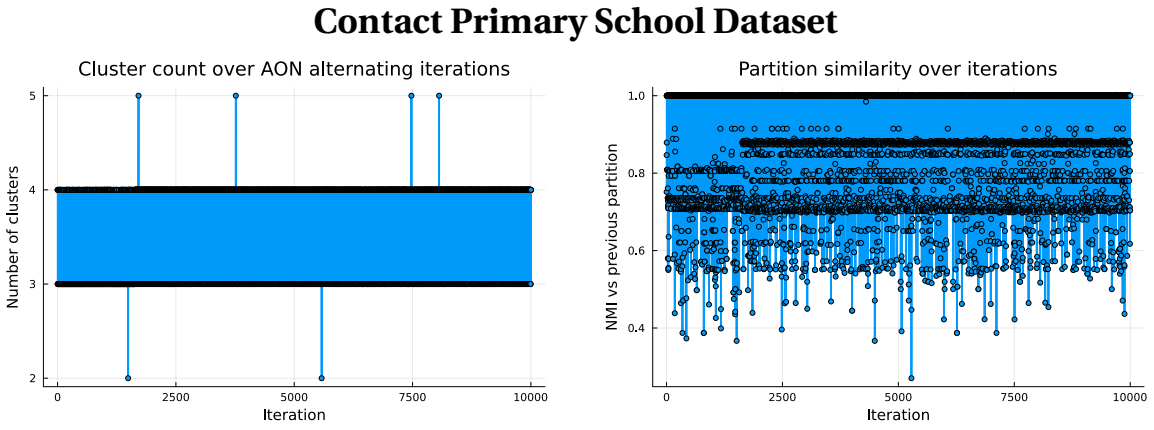
Figure 6.4 demonstrates the diminishing discovery rates towards zero. This means that as the iterations increase, the Louvain is more often returning back to partitions that it has already seen before. They do not show *which* of the partitions are being found more often, which could be a future area to look like. The rapidly decreasing rate does, however, suggest that the number of potentially-optimal partitions that



(a) Cluster counts per iteration.

(b) Consecutive NMI per iteration.

Figure 6.2: Partition exploration behavior of `AON_Louvain` across iterations on the Enron email dataset. **(a)** The number of clusters in each proposed partition oscillates among a small set of values, with occasional excursions to lower counts before returning. **(b)** Normalized mutual information between each proposed partition and the one immediately preceding it. Values below 1 indicate structurally distinct consecutive partitions found by the Louvain algorithm; the algorithm rarely produces identical partitions back-to-back, reflecting broad exploration of the partition space. Note that these partitions are those *proposed* by the Louvain step, not the partitions *selected* by the full alternating optimization pipeline.



(a) Cluster counts per iteration.

(b) Consecutive NMI per iteration.

Figure 6.3: Partition exploration behavior of `AON_Louvain` across iterations on the Contact Primary School dataset. **(a)** The number of clusters mainly oscillates between 3 and 4 clusters. **(b)** Normalized mutual information between each proposed partition and the one immediately preceding it. The graph shows strong variation across Louvain iterations, from 1 to less than 0.4.

the Louvain discovers is decreasing overtime. If it was continuing to find new, optimal partitions, we would expect to see a new, large upticks in the discovery rate as algorithm starts exploring the new space. In this paper we use a 15% learning rate cutoff for the Enron dataset, chosen partly due to runtime considerations, but this parameter can be adjusted and more sensitive clustering in the future could reduce this rate to a much lower number. Future work on convergence could also look at a mixture of learning rate and the duration at one partition to capture the combination between the probability of discovering a new admissible partition and how much the algorithm is currently moving between partitions. Seeing low rates in both of these categories would provide a convincing argument that the algorithm is done finding new, potentially-optimal partitions.

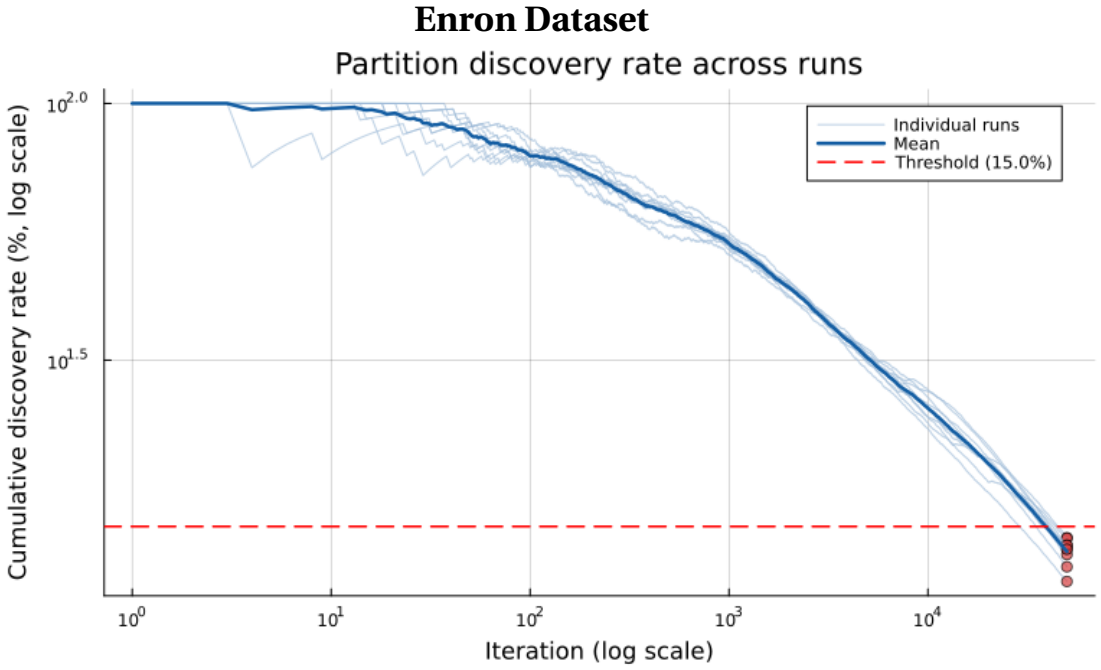


Figure 6.4: Cumulative discovery rates of 9 runs of 50,000 iterations each on log-scaled x and y axis. The rates rapidly diminish overtime, indicating a more-explored partition space than in earlier iterations.

The discovery rate in Figure 6.5 provides an even more promising picture of convergence. In this graph for the Contact Primary School dataset, the rates decline at almost a slope of -1 on the log-log scale. Since a slope of exactly -1 would mean that no new partitions are being found, anything with a similar slope indicates a very small number of new partitions being found. Clearly, the Primary School dataset has a community structure that converges to a low discovery rate much quicker than the Enron dataset; however, it also could provide foresight into what the Enron discovery rate behavior may eventually look like under enough iterations.

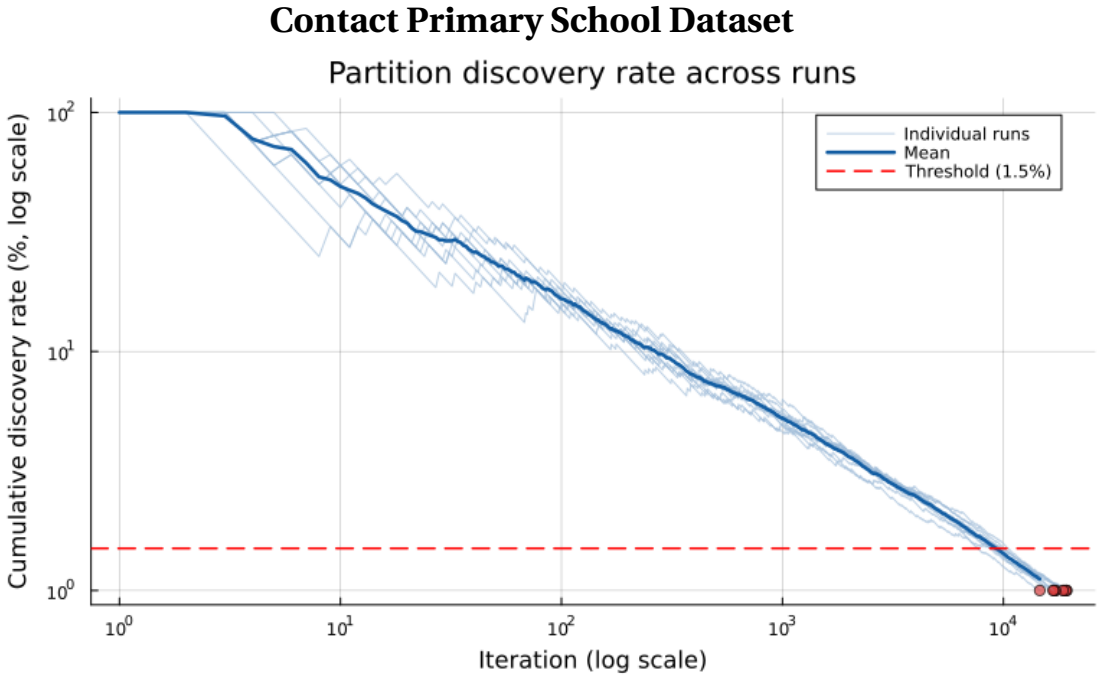


Figure 6.5: Cumulative discovery rates of 9 runs of 30,000 iterations each on log-scaled x and y axis. The rates rapidly diminish overtime, indicating a more-explored partition space than in earlier iterations.

At the same time, Figure 6.6 shows the amount of time throughout the algorithm that one point is consistently being mapped to during the check step. Each time a

new partition is mapped to, the line returns to zero. These spots are marked with a vertical line on the graph. Here, we can see there are multiple windows where the algorithm "dwells" on one partition before finding a new, more optimal one. It demonstrates that, despite dwelling on one spot for a long period of time, new ones consistently appear. This is further indication that study of the discovery rate of partitions is valuable, as it provides a more clear picture of the slowing pace of discovery and subsequent likelihood that a new fixed point will be found.

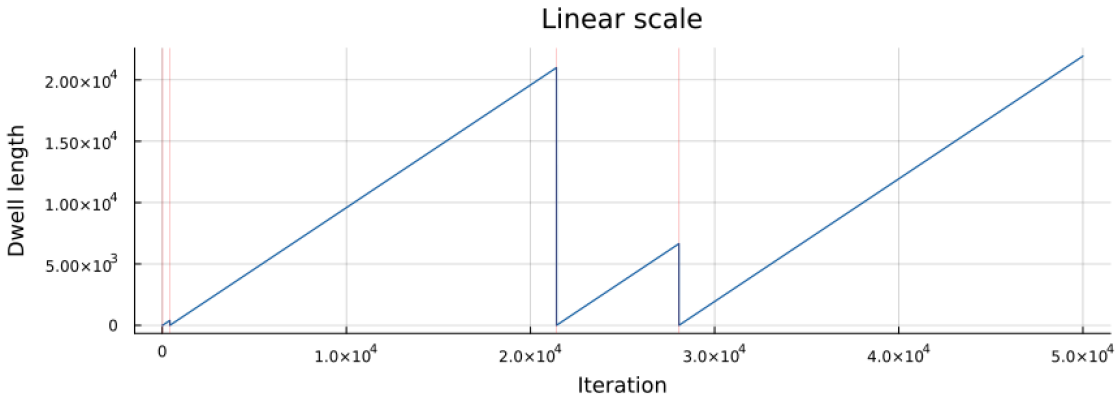


Figure 6.6: Dwell length at each parameter fixed point across alternating optimization iterations (linear scale) **for the Enron dataset**. The sawtooth pattern reflects periodic escapes to higher-modularity partitions, with dwell length resetting each time the optimizer discovers a new best solution in the partition space.

6.3 Cross-Run Similarity

In the Enron set, the full partition spaces explored across independent runs show substantial diversity. Among these runs, the highest pairwise similarity is only 55% (Figure 6.7a). That is, at most 55% of partitions in a run can be found from any other given run. The subset of "notable" partitions, which we define as any partition in the finite deterministic partition mapping that is *mapped to* by at least one other

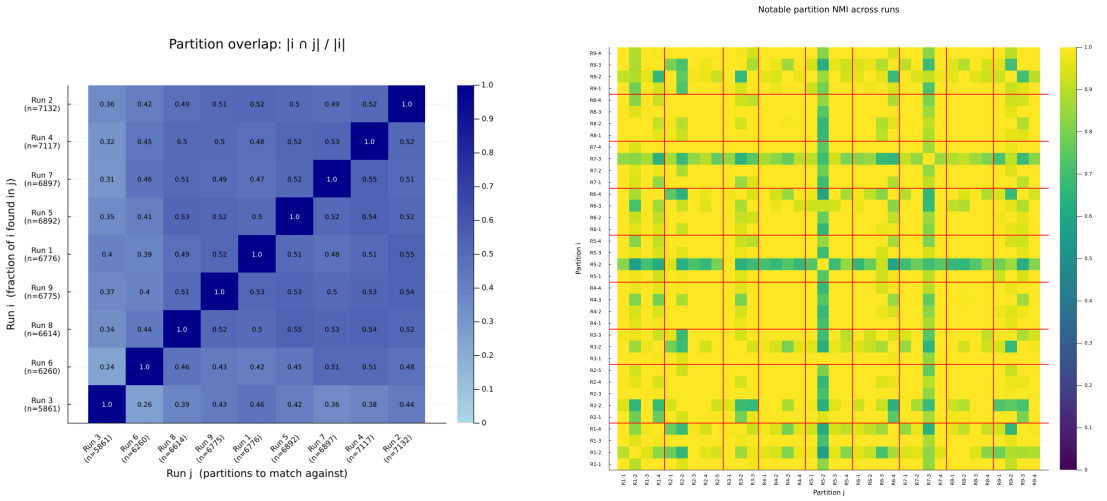
partition, shows striking cross-run consistency. Formally, we define a *notable partition* $z^* \in \Sigma$ if and only if there exists $z \in \Sigma$ such that $\sigma(z) = z^*$. As shown in Figure 6.7b, the majority of notable partitions compare to each other with NMI scores that are nearly 1. This indicates that, although the overall partitions may not be identical, the ones with structural meaning are consistent. Despite the stochasticity of the alternating optimization step, the algorithm is able to identify the same partitions as consistent attractors across runs.

This contrast between the two NMI comparisons is meaningful. The low pairwise overlap in Figure 6.7a supports the idea that the Louvain-based search explores a large and diverse region of the partition space. It also shows the randomness of the exploration phase and supports and the work in [25]. At the same time, a small number of structurally distinct outliers (notably R2-2 and R1-4, which are the teal blocks in Figure 6.7b) suggest there are other competing solutions, which could also be consistent with the known modularity degeneracy in [25].

In contrast, the primary school dataset showed much higher overlap between runs. Notably, the primary school's discovery rate was set at 1.5% rather than 15 for the Enron set. This higher overlap could indicate that the primary school's partition space was more completely explored compared to the Enron runs, which would make sense given the lower discovery rate. It could also be indicative of a simpler, easier-to-find community structure, yielding fewer near-optimal partitions findable by the Louvain algorithm. At the same time, the comparisons of the "notable" partitions yielded similar results. The primary school comparison also had the majority of its squares near 1.0, with some runs displaying distinct differences from the rest. This indicates that, even despite a lower cross-run similarity in the Enron set, the

consistency of admissible points across runs remains similar. This could provide meaningful insights into the importance of the discovery rate threshold and how many iterations are necessary for finding the same results. If the similarity of notable partitions is consistent across these two datasets, even if their cross-run overlap differs drastically poses additional questions as to when a viable stopping point may be.

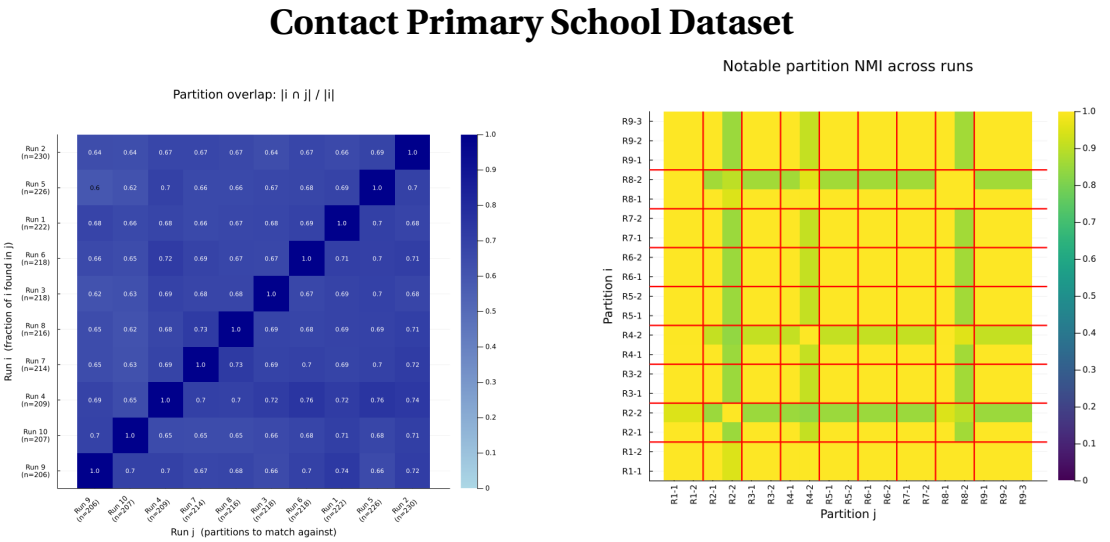
Enron Dataset



(a) Pairwise partition overlap $|i \cap j|/|i|$ across 9 independent runs (5,861–7,132 unique partitions per run). Off-diagonal values range from 0.24 to 0.55, demonstrating that runs explore non-overlapping partitions in the space.

(b) Pairwise NMI among notable partitions (in-degree ≥ 1 in the discrete map). Red lines delineate run boundaries. Despite the low run-level partition overlap shown in (a), notable partitions achieve high NMI across runs in the majority of cases, indicating convergence to the same community structures. A small number of structurally distinct partitions (e.g., R2-2, R1-4) represent candidate competing solutions.

Figure 6.7: Partition diversity versus attractor consistency. (a) The full partition spaces explored across runs are broadly non-overlapping, confirming stochasticity in the search. (b) Among notable partitions, NMI is high across runs, demonstrating that the mapping phase finds a consistent subset of community structures over different runs.

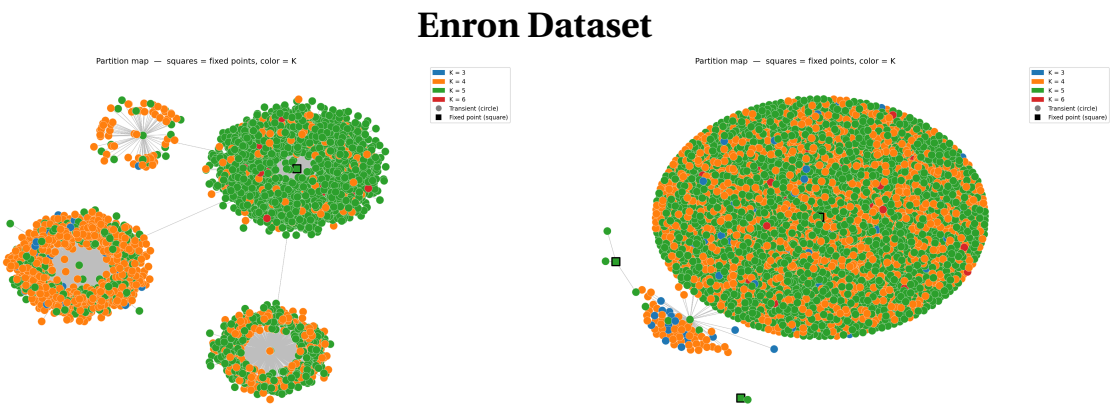


(a) Pairwise partition overlap across 10 runs (206–230 partitions per run; overlap range 0.60–0.74).

(b) Pairwise NMI among notable partitions across all 9 runs. Greater prevalence of green/teal blocks indicates more structural diversity among attractors relative to the Enron dataset.

Figure 6.8: Partition diversity and notable partition consistency for the Contact Primary School hypergraph. (a) Pairwise overlap $|i \cap j|/|i|$ across all partitions discovered per run. (b) Pairwise NMI among notable partitions (in-degree ≥ 1 in the discrete map). In contrast to Enron, runs show substantially higher partition overlap (0.60–0.74).

6.4 Finite Deterministic Maps of the AON Exploration

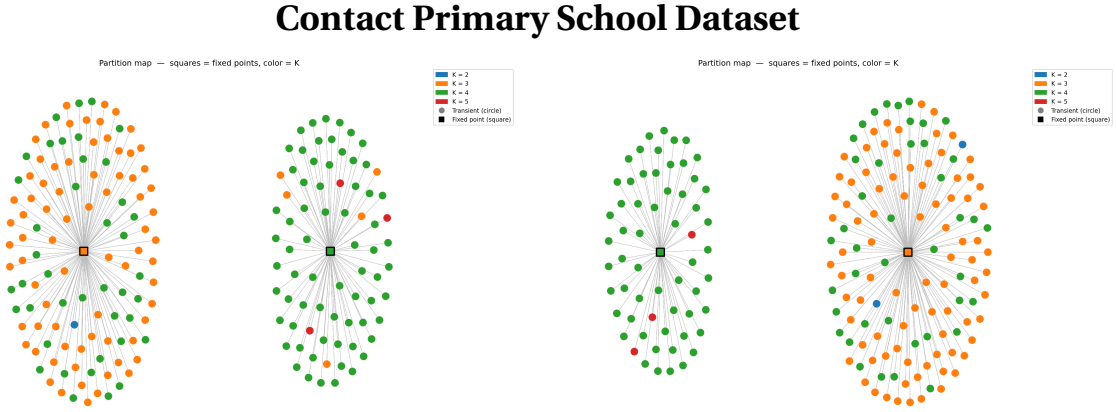


(a) Partition map for the first run of the exploration phase. Four distinct components are visible, with a single fixed point (black square, $K = 5$). The inter-component edges indicate that satellite components funnel into the main basin. Node color encodes cluster count $K \in \{3,4,5,6\}$; node size reflects in-degree.

(b) Partition map for the fourth run of the exploration phase. Three fixed points are identified (black squares, all $K = 5$).

Figure 6.9: Finite-state discrete maps of the AON alternating optimization pipeline across two hypergraph datasets. Each node represents a unique partition $\mathbf{z}$ discovered during iterative optimization; a directed edge $\mathbf{z}_i \to \mathbf{z}_j$ indicates that re-estimating $(\beta_k, \gamma_k, \omega_{k0}, \omega_{k1})$ under $\mathbf{z}_i$ and re-running AON-LOUVAIN yields $\mathbf{z}_j$. Fixed points ($\mathbf{z}^* = f(\mathbf{z}^*)$, squares) are self-consistent partitions in that they maximize AON modularity at their own parameters. In both runs, all fixed points have $K = 5$ communities, suggesting a preferred scale of community structure robust to initialization and parameter variation. The length of each edge is scaled by the *relative change in modularity* from the node's own modularity score to the improvement found by mapping to the next node.

Figure 6.9a shows the resulting finite deterministic map from the first alternating optimization exploration. The map forms four distinct components, the largest of which contains a single fixed point ($K = 5$, black square) embedded near its center with high in-degree, indicating that a large fraction of the partitions flow towards it in the mapping phase. That is, for the majority of partitions, this fixed point



(a) Partition map from a run 2 of the exploration phase.

(b) Partition map from a run 5 of the exploration phase.

Figure 6.10: Finite-state discrete maps of the AON alternating optimization pipeline across two hypergraph datasets. Each node represents a unique partition $\mathbf{z}$ discovered during iterative optimization; a directed edge $\mathbf{z}_i \to \mathbf{z}_j$ indicates that re-estimating $(\beta_k, \gamma_k, \omega_{k0}, \omega_{k1})$ under $\mathbf{z}_i$ and re-running AON-LOUVAIN yields $\mathbf{z}_j$. Fixed points ($\mathbf{z}^* = f(\mathbf{z}^*)$, squares) are self-consistent partitions in that they maximize AON modularity at their own parameters. The length of each edge is scaled by the *relative change in modularity* from the node's own modularity score to the improvement found by mapping to the next node.

achieves a higher modularity score under their optimal parameters. Three other clusters of components are also visible, each of which gather around a singular high-in-degree transient point and without a fixed point. This transient-hub phenomena indicates the algorithm found a partition that, for many iterations produced near-optimal partitions, but ultimately also mapped towards the dominant fixed point or basin. While the partition space explored spans $K \in \{3,4,5,6\}$, with K=4 and K=5 partitions co-occurring throughout all components, the sole fixed point has $K = 5$.

Since the distance between points denotes the relative improvement from a node's own modularity score to the one it maps to, it makes sens that nodes mapping to a given partition are clustered tightly around each other. The central discovery these plots show that there does not exist a single "best" partition. That is, they add an analysis of the space that goes beyond the 20-iteration methods in [1].

The map for the 4th alternating AON exploration run in the Enron set (Figure 6.9b) exhibits a similar, but slightly different structure from the run on the left. It contains a single dominant component containing one embedded fixed point ($K = 5$), plus two peripheral fixed points with near-zero in-degree. These peripheral points are significant in that they are fixed points, but they attract significantly less partitions than the dominant fixed point in the map. This could indicate either (a) the space around the partitions are more stable so the Louvain does not produce as many near-optimal partitions or (b) the point is less stable and small perturbations from these points, even if found from the fixed partitions, maps to the larger basin. This is also supported by the larger distance between clusters of nodes compared to the tighness of the clusters themselves. A small satellite of predominantly $K \in \{3,4\}$ partitions feeds into the main component rather than constituting an independent basin.

Notably, across both runs in the Enron set and despite broad exploration of partition space, *all fixed points have* $K = 5$ *communities.* It is also notable that there are multiple fixed points. This consistency is not imposed by the algorithm, but still emerges naturally through the exploration phase. This implies parameters under which K=5 partitions maximize AON modularity. This suggests that $K = 5$ represents a preferred and stable structuring of the communities in the network.

Figure 6.10 shows two examples of maps from the Contact Primary School dataset. These maps are much more similar to each other than the ones from the Enron dataset. This could, in part, be because the exploration phase for this dataset ended when the discovery rate was at 1.5% instead of 15%. Or, it could indicate that the community structure in this algorithm is much more apparent. Both maps show two distinct fixed points, one with 3 clusters and one with 4 clusters. Additionally, the transient points mapping to these partitions largely have the same number of clusters as the point they map to. This is not necessarily the case in the Enron examples. The fixed point with four communities has a tighter radius of partitions surrounding it compared to the fixed point with three communities. The two fixed points are notable because they show that there are clear, distinct optimal partitions that occupy different parts of the partition space. The finite deterministic maps are evidence of the rich partition space that cannot be captured exclusively through the original AON alternating algorithm.

Chapter 7

Conclusion

The alternating AON algorithm provides a meaningful connection to the dyadic graph case and develops the methods available in hypergraphs. This thesis extends the work of Chodrow et al. [1] and finds significant improvements in the partition mapping space. Specifically, it improves upon the current work in the following ways:

Numerical parameter stability The improved stability of the β and γ parameters throughout iteration is a clear demonstration of the increased correctness and stability of the algorithm. In addition to correcting the calculation of the γ vector, this work addresses edge cases in stability that can cause degeneracies in the space exploration.

Partition space exploration As demonstrated by the size of Σ, particularly in the Enron runs, many near-optimal partitions exist for a given network. This is consistent with the known exponential growth of near-optimal partitions in highly modular structures [25]. Tracking what the algorithm has already seen and re-evaluating each discovered partition under the most recently estimated parameters allows the algo-

rithm to leverage the full partition space. This gives the algorithm the opportunity to escape local optima and re-direct its path through the partition space by back-tracking to a more optimal spot in the trajectory. The work in Chodrow et al. [1] sets the iteration limit at 20 and picks the best likelihood score out of the batch of partitions. The exploration methods in this thesis expand beyond this framework, providing a rich method for exploring the space.

Convergence Given the greedy randomness of the Louvain algorithm, guaranteeing convergence among partitions is difficult. In this thesis we analyze potential measures for convergence of this algorithm. Specifically, we look at the among of time the algorithm spends at one partition (Figure 6.6) as well as the diminishing discovery rate (Figure 6.4 and 6.5). These figures show that the discovery rate does diminish over time. Further, Figures 6.7 and 6.8 provide more insight into the tradeoff between discovery rate and the algorithm's ability to find the important partitions in the space across runs.

Finite deterministic partition mapping This work builds off of Gibson and Mucha [15], which performs finite partition mapping in one dimension or two dimensions in the multilayer case. Here, we apply the same framework, but to $2(\bar{k}-1)$ dimensions, where $\bar{k}$ is the maximum edge size considered. In the original alternating algorithm, experiments run 20 iterations and take the highest-likelihood partition from among those found. Figures 6.9a and 6.9b demonstrate that the partition spaces are far richer than a fixed iteration budget can reliably characterize. By treating the set of discovered partitions Σ as a finite domain and constructing a deterministic map $\sigma : \Sigma \to \Sigma$, this thesis identifies fixed-point partitions and other important points.

This deterministic restriction phase transforms an inherently stochastic alternating procedure into a structural analysis of the partition space. As discussed in Section 3, graphs can exhibit multiple structures and community scales, making it important to have a more complete analysis of the explored partition space.

7.0.1 Limitations and Future Work

All formulas in this hypergraph framework operate under the "all-or-nothing" assumption, which limits the ability to describe and represent the nature of the node interactions. In this thesis, we chose datasets based on their size and the methods in [1], but the results may be limited by the scale of the AON interactions of the systems. Analysis of this work was also limited by time constraints and computing power. For example, the Enron dataset's discovery rate was set at 15% to allow for reasonable computing time over repeated runs. In practice, this rate may have been too high to fully explore the partition space.

In addition to addressing these limitations, the work in this thesis also provides many avenues for future exploration. First, this work only uses discovery rate as a threshold for convergence. It considers other measures, like partition "dwell" time, but determines the cutoff iterations strictly by the discovery rate in the exploration step. Additionally, since systems do not behave the same across datasets, this work should be repeated on more hypergraphs, potentially with stricter discovery rate thresholds. Since alternating modularity maximization exists in the regular graph case, comparisons to partitions found via regular graph abstraction and subsequent modularity maximization clustering could provide insights into the value gained from the use of the higher-order structures.

Future work could also address runtime complexity and efficiency of the algorithm. Assuming a constant calculation time for the Louvain and parameter estimation steps, the exploration phase currently runs at $O(n^2)$ complexity where n is the number of iterations. This is because the algorithm repeats n times and re-calculates the modularity score for all unique partitions each time, which could be as large as the current number of iterations. The runtime for this algorithm could be improved by storing modularity for past partitions, for example.

Hypergraphs have the potential to bring significant insights to real-world problems and extensions of this work could bring both practical and theoretical benefits to the field.

Bibliography

[1] Philip S. Chodrow, Nate Veldt, and Austin R. Benson. Generative hypergraph clustering: From blockmodels to modularity. *Science Advances*, 7(28): eabh1303, July 2021. ISSN 2375-2548. doi: 10.1126/sciadv.abh1303. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC11559555/.

[2] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, January 2020. ISSN 2666-6510. doi: 10.1016/j.aiopen.2021.01.001. URL https://www.sciencedirect.com/science/article/pii/S2666651021000012.

[3] D. DEVAKUMAR, A. KITCHING, D. ZENNER, A. TOSTMANN, and M. MELTZER. Tracking sickness through social networks – the practical use of social network mapping in supporting the management of an E. coli O157 outbreak in a primary school in London. *Epidemiology and Infection*, 141(10): 2022–2030, October 2013. ISSN 0950-2688. doi: 10.1017/S0950268813000344. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC3757920/.

[4] Yujia Han, Nigel David Caldwell, and Abhijeet Ghadge. Social network analysis in operations and supply chain management: a review and revised research

agenda. *International Journal of Operations & Production Management*, 40(7-8):1153–1176, June 2020. ISSN 0144-3577. doi: 10.1108/IJOPM-06-2019-0500. URL https://doi.org/10.1108/IJOPM-06-2019-0500.

[5] Caroline Haythornthwaite. Social network analysis: An approach and technique for the study of information exchange. *Library & Information Science Research*, 18(4):323–342, September 1996. ISSN 0740-8188. doi: 10.1016/S0740-8188(96)90003-1. URL https://www.sciencedirect.com/science/article/pii/S0740818896900031.

[6] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 103(23): 8577–8582, June 2006. ISSN 0027-8424. doi: 10.1073/pnas.0601602103. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC1482622/.

[7] Yang Xu, Arun Srinivasan, and Lingzhou Xue. A Selective Overview of Recent Advances in Spectral Clustering and Their Applications. In Yichuan Zhao and (Din) Ding-Geng Chen, editors, *Modern Statistical Methods for Health Research*, pages 247–277. Springer International Publishing, Cham, 2021. ISBN 978-3-030-72437-5. doi: 10.1007/978-3-030-72437-$5_12.URL$https://doi.org/10.1007/978-3-030-72437-5_12https://doi.org/10.1007/978-

[8] Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, June 1983. ISSN 03788733. doi: 10.1016/0378-8733(83)90021-7. URL https://linkinghub.elsevier.com/retrieve/pii/0378873383900217.

[9] Haifa Gmati, Amira Mouakher, and Inès Hilali-Jaghdam. B-CD: Com-

munity Detection in Bipartite Networks. *Procedia Computer Science*, 159: 313–322, January 2019. ISSN 1877-0509. doi: 10.1016/j.procs.2019.09.186. URL https://www.sciencedirect.com/science/article/pii/S1877050919313687.

[10] Charalampos E. Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable Motif-aware Graph Clustering. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 1451–1460, Republic and Canton of Geneva, CHE, April 2017. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4913-0. doi: 10.1145/3038912.3052653. URL https://dl.acm.org/doi/10.1145/3038912.3052653.

[11] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: application in VLSI domain. In *Proceedings of the 34th annual Design Automation Conference*, DAC '97, pages 526–529, New York, NY, USA, June 1997. Association for Computing Machinery. ISBN 978-0-89791-920-3. doi: 10.1145/266021.266273. URL https://dl.acm.org/doi/10.1145/266021.266273.

[12] Ju Niu and Yuhui Du. Applications of hypergraph-based methods in classifying and subtyping psychiatric disorders: a survey. *Radiology Science*, 02(01):83–95, December 2023. doi: 10.15212/RADSCI-2023-0008. URL https://mednexus.org/doi/10.15212/RADSCI-2023-0008.

[13] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. HyperGCN: A New Method For Training Graph Convolutional Networks on Hypergraphs. In *Advances in Neural Infor-*

mation Processing Systems, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/1efa39bcaec6f390014916069369453

[14] M. E. J. Newman. Equivalence between modularity optimization and maximum likelihood methods for community detection. *Physical Review E*, 94(5):052315, November 2016. doi: 10.1103/PhysRevE.94.052315. URL https://link.aps.org/doi/10.1103/PhysRevE.94.052315.

[15] Ryan A. Gibson and Peter J. Mucha. Finite-state parameter space maps for pruning partitions in modularity-based community detection. *Scientific Reports*, 12(1):15928, September 2022. ISSN 2045-2322. doi: 10.1038/s41598-022-20142-6.

[16] Kenji NAKAMURA. Technical Report: Serious Doubt on the Authenticity of the Enron Email Corpus. Technical report, Zenodo, April 2026. URL https://zenodo.org/doi/10.5281/zenodo.19425640.

[17] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon Kleinberg. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences*, 115(48): E11221–E11230, November 2018. doi: 10.1073/pnas.1800683115. URL https://www.pnas.org/doi/full/10.1073/pnas.1800683115.

[18] Valerio Gemmetto, Alain Barrat, and Ciro Cattuto. Mitigation of infectious disease at school: targeted class closure vs school closure. *BMC Infectious Diseases*, 14(1):695, December 2014. ISSN 1471-2334. doi: 10.1186/s12879-014-0695-9. URL https://doi.org/10.1186/s12879-014-0695-9.

[19] Juliette Stehlé, Nicolas Voirin, Alain Barrat, Ciro Cattuto, Lorenzo Isella,

Jean-François Pinton, Marco Quaggiotto, Wouter Van den Broeck, Corinne Régis, Bruno Lina, and Philippe Vanhems. High-Resolution Measurements of Face-to-Face Contact Patterns in a Primary School. *PLOS ONE*, 6(8): e23176, August 2011. ISSN 1932-6203. doi: 10.1371/journal.pone.0023176. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0023176.

[20] SocioPatterns – SocioPatterns, . URL https://sociopatterns.org/.

[21] Sameer Agarwal, Kristin Branson, and Serge Belongie. Higher order learning with graphs. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 17–24, New York, NY, USA, June 2006. Association for Computing Machinery. ISBN 978-1-59593-383-6. doi: 10.1145/1143844.1143847. URL https://dl.acm.org/doi/10.1145/1143844.1143847.

[22] Aviad Rubinstein. Detecting communities is hard, and counting them is even harder, November 2016. URL https://arxiv.org/abs/1611.08326v1.

[23] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, June 2002. doi: 10.1073/pnas.122653799. URL https://www.pnas.org/doi/10.1073/pnas.122653799.

[24] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, October 2008. ISSN 1742-5468. doi: 10.1088/1742-5468/2008/10/P10008. URL https://doi.org/10.1088/1742-5468/2008/10/P10008.

[25] Benjamin H. Good. Performance of modularity maximization in practical contexts. *Physical Review E*, 81(4), 2010. doi: 10.1103/PhysRevE.81.046106.

[26] Vincent Traag, Ludo Waltman, and Nees Jan van Eck. From Louvain to Leiden: guaranteeing well-connected communities, October 2018. URL https://arxiv.org/abs/1810.08473v3.

[27] Austin R. Benson, David F. Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295): 163–166, July 2016. doi: 10.1126/science.aad9029. URL https://www.science.org/doi/10.1126/science.aad9029.

[28] Philip Chodrow. Configuration models of random hypergraphs. *Journal of Complex Networks*, 8, June 2020. doi: 10.1093/comnet/cnaa018.

[29] Debarghya Ghoshdastidar and Ambedkar Dukkipati. Consistency of Spectral Hypergraph Partitioning Under Planted Partition Model. *The Annals of Statistics*, 45(1):289–315, 2017. ISSN 0090-5364. URL https://www.jstor.org/stable/44245779.

[30] Bogumił Kamiński, Paweł Misiorek, Paweł Prałat, and François Théberge. Modularity based community detection in hypergraphs. *Journal of Complex Networks*, 12(5):cnae041, September 2024. ISSN 2051-1329. doi: 10.1093/comnet/cnae041. URL https://doi.org/10.1093/comnet/cnae041.

[31] nveldt. HyperModularity.jl/src/AON_hyperlouvain.jl at 71515db80150e59eaf211103bccedceac0ef4e29 · nveldt/HyperModularity.jl. URL https://github.com/nveldt/HyperModularity.jl/blob/71515db80150e59eaf211103bcce

[32] Relating Modularity Maximization and Stochastic Block Models in Multilayer Networks, . URL https://epubs.siam.org/doi/epdf/10.1137/18M1231304.

[33] Niko Motschnig, Alexander Ramharter, Oliver Schweiger, Philipp Zabka, and Klaus-Tycho Foerster. On Comparing and Enhancing Two Common Approaches to Network Community Detection. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Madrid, Spain, December 2021. IEEE. ISBN 978-1-7281-8104-2. doi: 10.1109/GLOBECOM46510.2021.9685248. URL https://ieeexplore.ieee.org/document/9685248/.

Appendix

The following figures display the partition maps that were not pictured in the main body of this text.

Partition Maps for the Enron Email Dataset

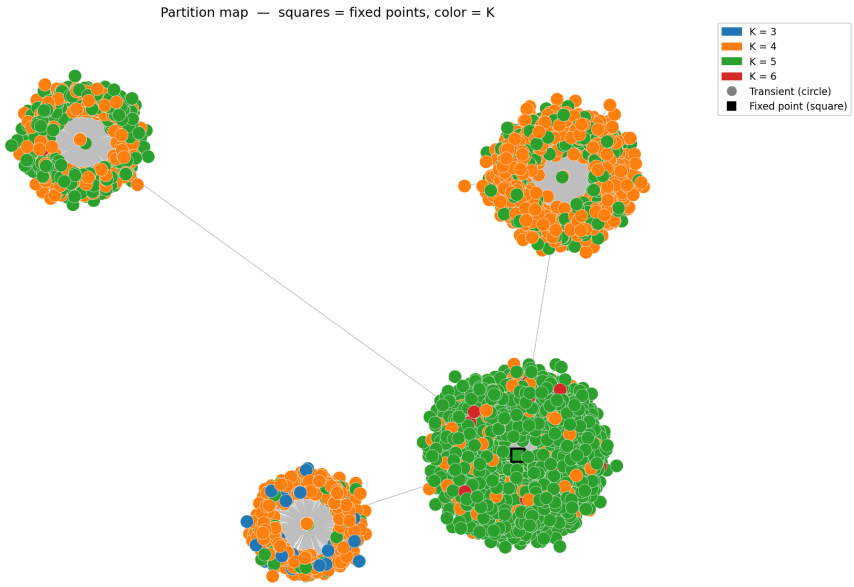


Figure 1: Partition map for run #2 of the Enron dataset. All methods follow directly from the figures described in Figure 6.9.

Figure 2: Partition map for run #3 of the Enron dataset. All methods follow directly from the figures described in Figure 6.9.

Figure 3: Partition map for run #5 of the Enron dataset. All methods follow directly from the figures described in Figure 6.9.

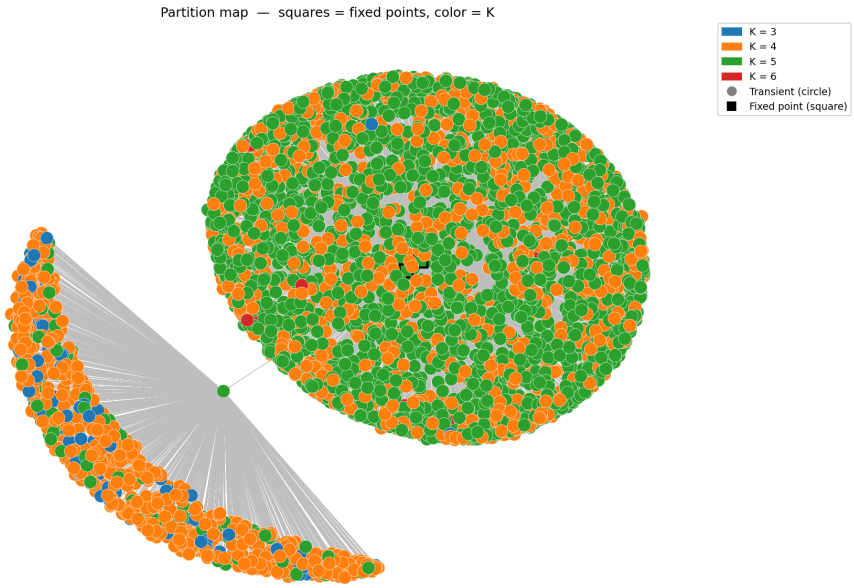


Figure 4: Partition map for run #6 of the Enron dataset. All methods follow directly from the figures described in Figure 6.9.

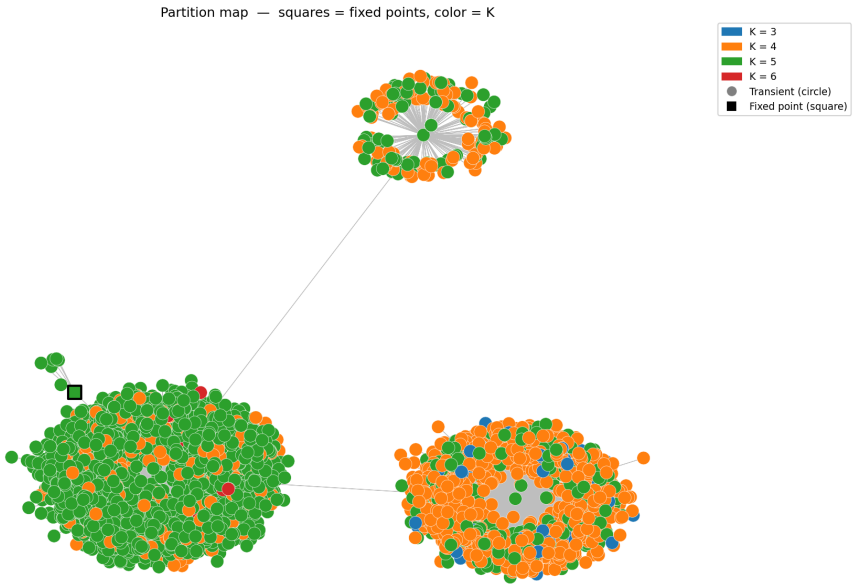


Figure 5: Partition map for run #7 of the Enron dataset. All methods follow directly from the figures described in Figure 6.9.

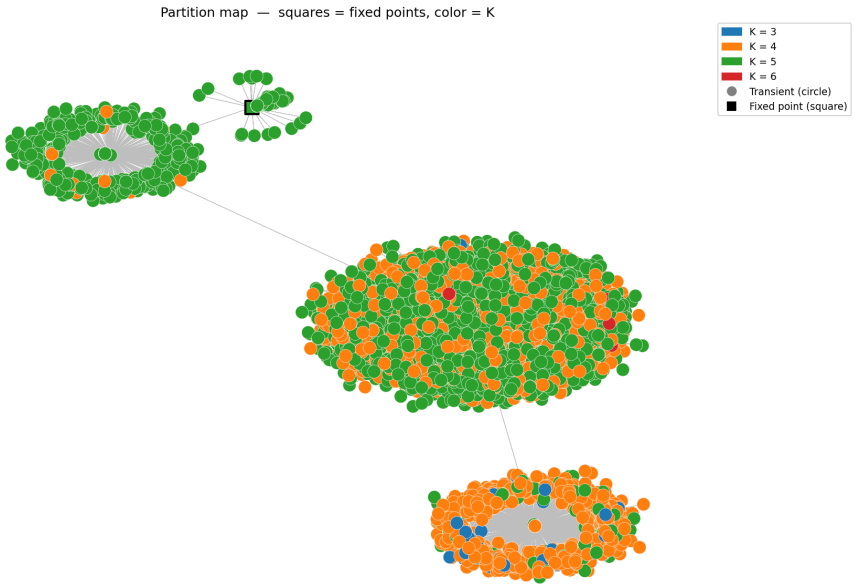


Figure 6: Partition map for run #8 of the Enron dataset. All methods follow directly from the figures described in Figure 6.9.

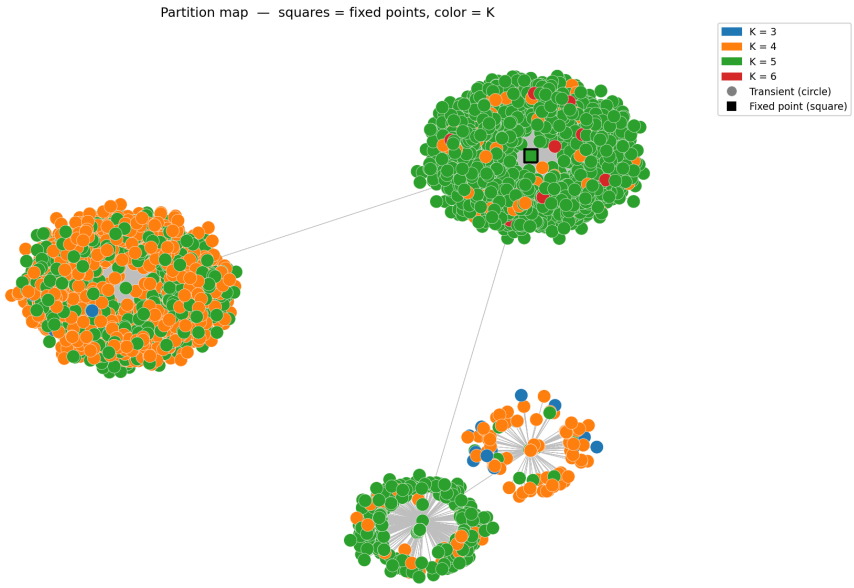


Figure 7: Partition map for run #9 of the Enron dataset. All methods follow directly from the figures described in Figure 6.9.

Partition Maps for the Contact Primary School Dataset

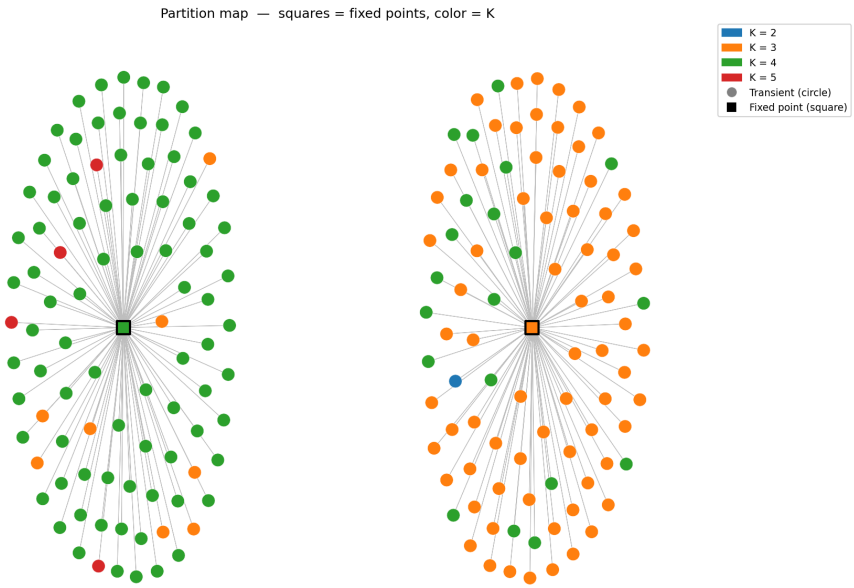


Figure 8: Partition map for run #1 of the CPS dataset. All methods follow directly from the figures described in Figure 6.10.

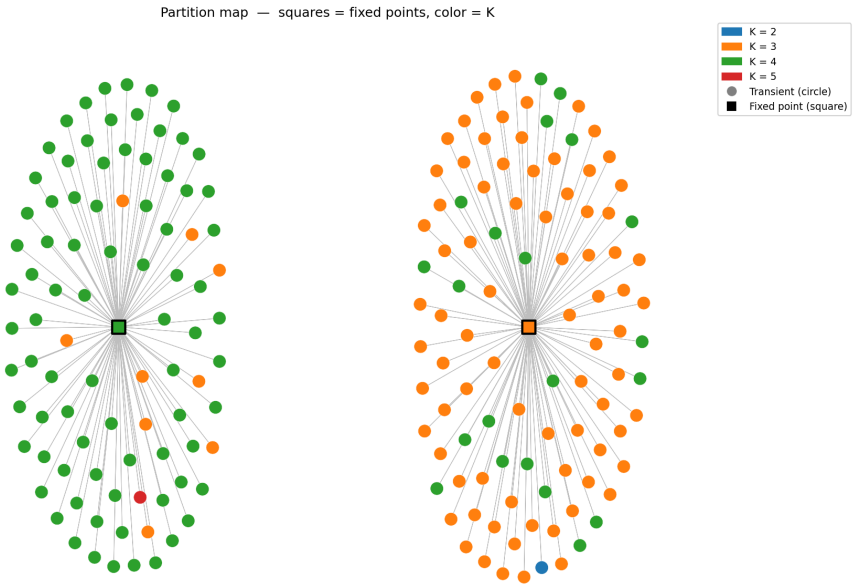


Figure 9: Partition map for run #3 of the CPS dataset. All methods follow directly from the figures described in Figure 6.10.

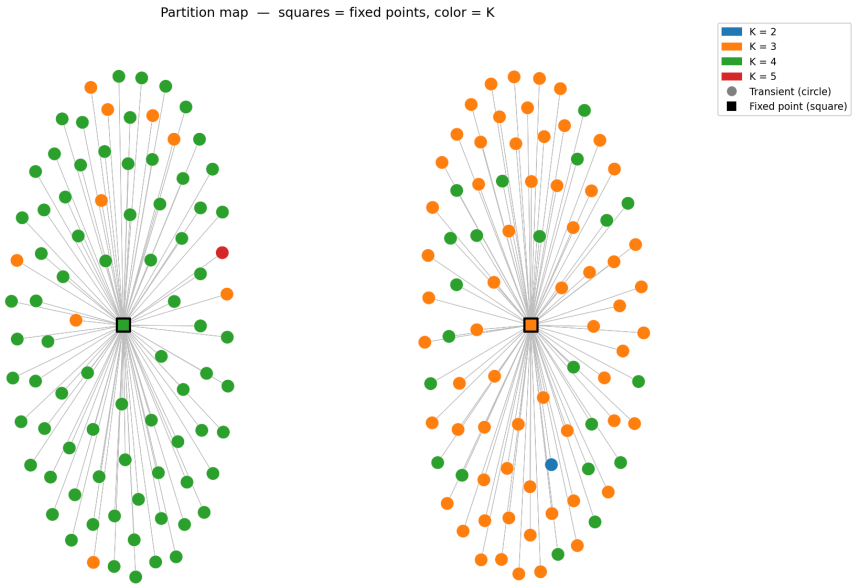


Figure 10: Partition map for run #4 of the CPS dataset. All methods follow directly from the figures described in Figure 6.10.

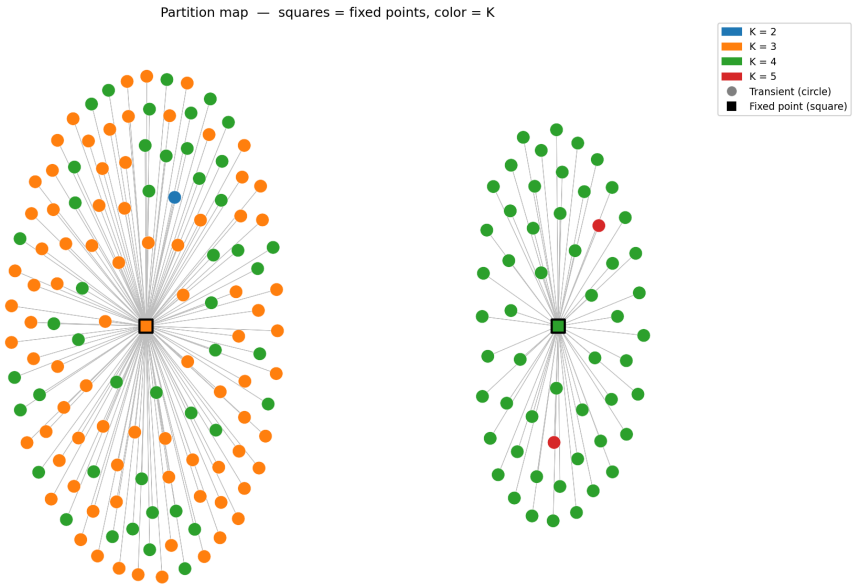


Figure 11: Partition map for run #6 of the CPS dataset. All methods follow directly from the figures described in Figure 6.10.

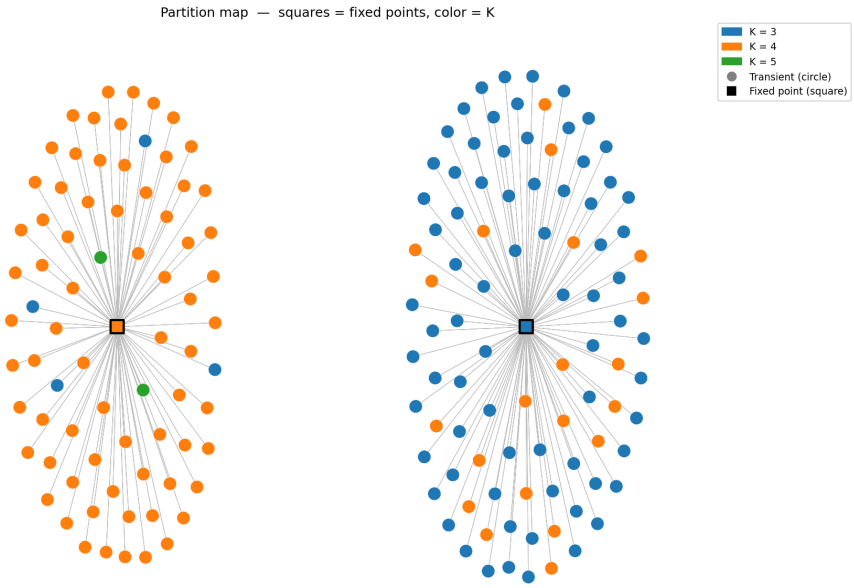


Figure 12: Partition map for run #7 of the CPS dataset. All methods follow directly from the figures described in Figure 6.10.

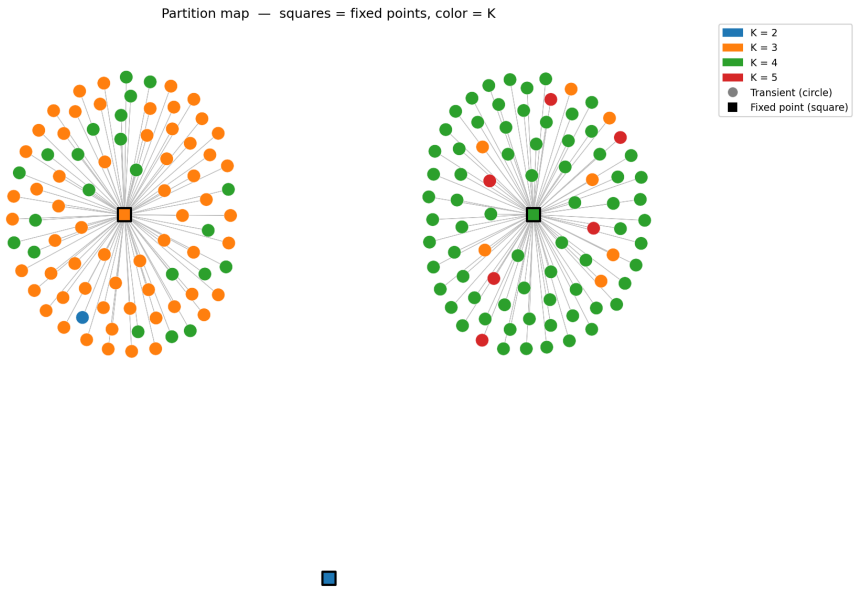


Figure 13: Partition map for run #8 of the CPS dataset. All methods follow directly from the figures described in Figure 6.10.