Graphs in Unsupervised Machine Learning

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
M. Sc. Solveig Peter
aus Ditzingen

Tübingen
2024

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	21.01.2025
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatterin:	Prof. Dr. Ulrike von Luxburg
2. Berichterstatter:	Prof. Dr. Robert Williamson

Abstract

Graphs are versatile data structures that represent individual data points and the intricate relationships between them, making them invaluable for modeling complex real-world systems. Their flexibility allows them to be applied to a wide range of problems, and with appropriate algorithms, they have become crucial in many domains. In particular, unsupervised machine learning – where patterns are discovered without labeled data – has greatly benefited from integrating graphs. This thesis presents an introduction to unsupervised machine learning on graphs, focusing on clustering and representation learning. We explore the strengths and challenges of applying graphs to machine learning, emphasizing these two specific tasks. Our contributions span two distinct perspectives within unsupervised learning on graphs. In the first part, we propose a novel clustering method inspired by the concept of tangles from mathematical graph theory. Tangles are used to detect densely connected subsets within a graph, and we adapt this idea to cluster data across various domains. Our approach yields a soft clustering hierarchy that is often inherently interpretable. We provide theoretical performance guarantees across different data types and validate the method's effectiveness through extensive empirical evaluations. The second part of this thesis delves into representation learning on graphs. We investigate the inductive biases of graph auto-encoders, connecting their non-linear architecture to a linear model. Our empirical findings show that linear models can match or surpass the performance of non-linear graph auto-encoders when node features are effectively leveraged. Additionally, we analyze the constraints imposed by features on the solution space, offering theoretical and empirical evidence that these features are the key to achieving strong performance in this setting.

Zusammenfassung

Graphen sind vielseitige Datenstrukturen, die einzelne Datenpunkte sowie die komplexen Beziehungen zwischen jenen abbilden. Durch ihre Vielseitigkeit sind sie für die Modellierung komplexer realer Systeme unverzichtbar geworden und dank ihrer Flexibilität können sie auf eine Vielzahl von Problemen angewendet werden. Sie sind, kombiniert mit geeigneten Algorithmen, in vielen Bereichen von entscheidender Bedeutung. Insbesondere profitiert das unüberwachte maschinelle Lernen – bei welchem Muster ohne gelabelte Daten detektiert werden – erheblich von der Integration von Graphen. Diese Arbeit bietet eine Einführung in das unüberwachte maschinelle Lernen auf Graphen, mit dem Schwerpunkt auf Clustering und Repräsentationslernen. Wir elaborieren die positiven und negativen Aspekte der Verwendung von Graphen im maschinellen Lernen und konzentrieren uns dabei auf die beiden eben genannten Aufgaben. Unser Beitrag umfasst zwei unterschiedliche Perspektiven auf das unüberwachte Lernen auf Graphen. Im ersten Teil stellen wir eine neuartige Clustering-Methode vor, die vom Konzept eines "Tangles" aus der mathematischen Graphentheorie inspiriert ist. Tangles dienen dazu, dicht verbundene Teilmengen innerhalb eines Graphen zu identifizieren, und wir adaptieren diese Idee, um Daten aus verschiedenen Domänen zu clustern. Unser Ansatz führt zu einer weichen Clustering-Hierarchie, die oft von Natur aus interpretierbar ist. Wir präsentieren theoretische Garantien für verschiedene Datentypen und validieren die Effektivität der Methode durch umfangreiche empirische Evaluierungen. Der zweite Teil dieser Arbeit widmet sich dem Repräsentationslernen auf Graphen. Wir untersuchen die induktiven Vorannahmen von Graph-Autoencodern und bringen die nichtlineare Architektur mit einem linearen Modell in Verbindung. Unsere empirischen Ergebnisse zeigen, dass lineare Modelle die Leistung nichtlinearer Graph-Autoencoder erreichen oder sogar übertreffen können, wenn Knoteneigenschaften effektiv genutzt werden. Zudem analysieren wir, wie der Lösungsraum durch Verwendung von Knoteneigenschaften beschränkt wird und liefern theoretische sowie empirische Belege dafür, dass genau diese Merkmale der Schlüssel zu guter Leistung und Generalisierung in diesem Kontext sind.

Acknowledgements

I would like to express my gratitude to my supervisor, Ulrike von Luxburg, for her scientific and endless emotional support throughout my PhD journey, especially during the challenging times of the COVID-19 pandemic, but also anytime actually. Your guidance extended beyond academic matters and was a constant source of encouragement through many life circumstance that tested me.

I also thank my second supervisor, Bob Williamson for the engaging conversations, which always left me feeling inspired and somehow more confident.

Ein herzliches Dankeschön geht an meine Mama, die jeden von mir gewählten Weg unterstützt und immer an mich geglaubt hat. Einen großen Teil meines Erfolgs verdanke ich dir, deinem Vertrauen in mich und unseren endlosen Telefonaten. “Auf all meinen Wegen hast du mich beschützt. Ich hoffe ich gebe dir das irgendwie zurück”. - Madsen

I want to thank my sister for always having an open ear and offering her kind, understanding nature during this time. Your support and compassion never go unnoticed.

I am grateful to Luca, Leena, Moritz, Michael, Sebastian, and all my colleagues from the TML group. The familiar atmosphere you fostered, along with the open doors for both scientific and personal discussions, made this journey so much more joyful.

A special thank you to my friends from the University and the MPI: Ben, Chris, Mila, and Rabanus. Thank you for listening to and sharing your own struggles. While this might seem minor to you, not feeling alone in this process was a huge part of my success.

A heartfelt thank you goes to Lara, my good friend and part-time roommate. Your supportive and understanding presence was invaluable. Our discussions were a constant source of comfort, and your help in taking care of Nami from time to time was a great support in balancing work and life.

Finally, my deepest thanks go to Janis, my husband, the father of my daughter, my partner in everything, and, above all, my very best friend. Thank you for celebrating every achievement with me, for holding every frustration, for every pep talk, every motivational word, for telling me to rest and for kicking my ass when necessary, and for countless well-needed hugs. Your support makes all the difference.

Contents

Chapter 1

Introduction

In this thesis, we will discuss graphs in unsupervised machine learning. This topic is very broad, and this thesis covers only a small part of it all. Our contributions are within the field of graph clustering and graph representation learning. The following introduction will focus on those two tasks. This thesis is structured in the following way: We introduce the benefits and drawbacks of graphs as a data structure in machine learning in Section 1.1. We discuss the general concept of unsupervised machine learning on graphs in Section 1.2. We discuss clustering and representation learning on graphs in Sections 1.2.1 and 1.2.2 and go into more detail on representation learning with graph neural networks in Section 1.2.3. We specifically discuss representation learning for clustering in Section 1.2.4 and finally shortly summarize the contributions in Section 1.3, which will be included in Chapter 2. We conclude this work in Chapter 3.

1.1 Graphs in Machine Learning

A graph is a powerful data structure. Besides the bits of information it contains for each data point, a graph can encode complex relationships and dependencies between those entities. When first introduced in the paper "Das Königsberger Brückenprolem" in Euler (1736), the former was even less relevant, and a graph would mostly be used for modeling relationships. The properties of the individual entities played a minor role. Euler modeled the problem of finding a path through Königsberg (now Kaliningrad, Russia) that goes over all seven of its bridges exactly once. He showed that it was impossible to do so and introduced the ideas of graph traversal and connected components. His analysis laid the foundations for modern graph theory and topology. Today, we call our street systems road networks and model them using graphs and graph algorithms to solve all sorts of problems, like finding the quickest route from A to B. This approach has become so prevalent that graphs are now used to model almost everything. Skimming through graph-learning papers, you will often find the abstract emphasizing graphs as a universal language for describing complex systems. Along

the lines of: "anything can be a graph," scientists model most real-world problems as graphs. Not only in the field of computer science but also in linguistics, biology, social science, physics, chemistry, and mathematics, graphs are leveraged to analyze data and solve complex problems. Researchers have even applied graph-based techniques to images, using spectral clustering for tasks like image segmentation – though newer algorithms have proven more efficient. While traditional machine learning algorithms have led to significant advancements, certain fields have benefited greatly from the application of machine learning specifically on graphs. In 2020, the first drug discovered using graph neural networks entered clinical trials[1]. Graph neural networks aid the constant progress in material science Reiser et al. (2022), helping the design of more efficient batteries Sarkar et al. (2014); Wu et al. (2018) or the discovery of new materials for superconductors Quinn and McQueen (2022).

In the following section, we introduce the general concept of graph learning before discussing the benefits and drawbacks of graphs in machine learning.

1.1.1 Graph Learning

Graph learning is a branch of machine learning that focuses on the analysis and interpretation of data represented in graph form. A graph $G = (V, E)$ consists of nodes V and links between those nodes E. Every entity $v_i \in V$ captures properties of the node. This can be of an arbitrary datatype. Every link e_{ij} captures the relationship between the two nodes v_i and v_j. This again can be arbitrary; edges could be categorical types of relationships or real values to represent the connectivity. In many applications, every node is represented by a d-dimensional feature vector $v_i \in \mathbb{R}^d$, and each edge is a single value $e_{ij} \in \mathbb{R}$. In this simple form, a graph is often represented by an adjacency matrix $A \in \mathbb{R}^{n \times n}$ and a feature matrix $X \in \mathbb{R}^{n \times d}$. Both representations are equivalent with $A[i, j] = e_{ij}$ and $X[i, :] = x_i$. There are other forms to capture the structural information of a graph, often derived from the adjacency matrix. A^k, for example, captures not only the dependencies of nodes connected by a link but also contains the k-hop relationship of nodes. If the graph is unweighted, that is, all values in A are 1, $A^k[i, j]$ counts the number of paths between v_i and v_j of length k. Another form is the Laplace matrix of a graph. It exists in several forms and is a combination of the adjacency matrix A and the degree matrix D: $L = D - A$, where $D[i, i] = \deg(n_i) = \sum_{j=1}^{n} A[i, j]$ aggregates the connectivity to all nodes connected to v_i. Captured in any way, graph learning leverages the structural information of a graph to learn and make predictions about either the individual nodes, edges, or the graph as a whole.

In the context of machine learning, leveraging the inherent properties of graphs often requires introducing inductive biases, that is assumptions about the data that enable models to learn effectively. For example, many algorithms on graphs implicitly assume

[1] DSP-1181: drug created using AI enters clinical trial
https://www.europeanpharmaceuticalreview.com/news/112044/dsp-1181-drug-created-using-ai-enters-clinical-trials/ Accessed: 2024-10-05

that nodes that are connected by a link are more similar to each other than nodes that are not connected by a link (homophily). These biases are crucial for guiding models to capture the relational and structural information within graph data. However one should be aware of the bias introduced into a pipeline, as it usually influences how the results need to be interpreted. Many very different techniques have been developed to analyze a graph, like clustering, representation learning, and node and link classification and prediction. We will discuss clustering and representation learning in Section 1.2.1 and Section 1.2.2 respectively. Besides traditional methods, graph neural networks have gained increasing attention in several areas of graph learning. They extend the concept of classical artificial neural networks to handle graph-structured data. We introduce the basic principles of graph neural networks in Section 1.2.3.

The power of graphs.

We observe that graphs can be used to model a variety of different data types. With nodes and links of arbitrary type, they represent complex relationships and can be highly irregular. This attributes graphs with a high expressive power. Different from most other data structures, graphs can handle heterogeneous data, where various types of nodes and edges coexist, such as in a knowledge graph where nodes represent a variety of real-world objects, events, concepts, etc., and links can express relationships, implications, dependencies, and so on. They can even model dynamic systems where nodes and edges evolve over time, enabling time-aware analyses in areas like epidemiology, where the spread of diseases can be modeled as a dynamic graph.

Given the flexibility of the structure, graphs can model a diverse range of real-world systems. With suitable algorithms, this tackles a bunch of relevant problems.

Graphs are used to model recommendation systems. One example is movie recommendation, where interaction data such as movie ratings can be represented by a graph where nodes represent the users and the movies, and the edges represent the observed ratings Van Den Berg et al. (2017). Modeled like this, link prediction algorithms are used to estimate the value of links that are not present. Similarly, graphs can model, and recommend objects in (online) stores, music on streaming platforms, websites for search engines, and even persons in social networks Wu et al. (2019).

In road networks, graphs are commonly used to model and forecast traffic flow Guo et al. (2019); Jiang et al. (2023). Temporal graphs capture recent, daily, or weekly dependencies. Each node captures the flow at various points in the past, and neural networks are employed to predict future flows at these nodes.

In chemistry and biology, molecules can be modeled as graphs, where nodes represent atoms and edges represent chemical bonds. Machine learning has been used to predict chemical reaction properties Do et al. (2019) or side effects for the combination of different drugs Zitnik et al. (2018). Rhee et al. (2018) use graph convolutional filtering to classify breast cancer.

These are just a few of the applications for which graphs are used today. However, if we do not gain any signal by adding structural information, modeling data as a graph might make the data more difficult to work with and a problem more difficult to solve.

The challenge with graphs.

Most of their positive attributes make graphs a specifically difficult data structure for analysis and machine learning.

As a key property, graphs consist of two types of data: nodes and edges. This introduces more complexity compared to simpler data structures like arrays or matrices. In addition, these nodes and edges can have both, arbitrary and also different types and attributes, node- and edge-wise. As a result, unlike text, audio, and images, graph data needs to be represented by irregular domains, making some essential operations of existing machine learning algorithms unsuitable. For example, matrix operators, convolutions, and pooling are not straightforwardly applicable to irregular data like graphs.

The nodes in a graph do not have a natural order. However, no matter in which order we present the nodes to the algorithm, for the same graph, the output of our algorithm needs to be the same. This means machine learning models need to be permutation invariant or be able to learn an order-invariant representation.

We usually also want the output of our algorithm of two different graphs to be different. However, determining whether two graphs are the same (isomorphic) or not is a non-trivial problem that graph neural networks, for example, can not solve Xu et al. (2018). This again stresses the challenging nature of graphs.

As often in machine learning, scalability can also be an issue for graphs. Some graphs, like social networks, can have millions of nodes and edges, and the nodes and edges themselves might be high-dimensional. Analyzing, processing, and storing graph data is often expensive and sometimes infeasible.

As a result, a whole branch of graph learning considered the problem of graph embeddings or representation learning. Network representation learning was introduced in recent years with the target of learning latent features of graph nodes with a low dimensional representation, usually in the Euclidean space (Grover and Leskovec, 2016; Sun et al., 2020; Yu et al., 2020). Based on this new representation, conventional machine learning methods can be employed to analyze the graph and discover hidden patterns in the data.

Graph representation learning is usually an unsupervised machine learning method. People argue it imposes as little inductive bias into the data as possible while embedding it into another space and dimension. The intuition behind this argument is that without labels, one introduces less guidance in learning with the hope of discovering any present structure in the data without restriction. However, all algorithms impose an inductive bias, and in many cases, it is crucial to understand these hidden assumptions for proper analysis. We will discuss biases in graph representation learning in more detail in the

second part of Chapter 2. First, we give a general overview of unsupervised machine learning, the motivation behind it, and current challenges in the following section.

1.2 Unsupervised Machine Learning on Graphs

Unsupervised machine learning is a branch of machine learning where models learn from data without explicit labels or predefined outcomes. The primary objective is to discover hidden patterns, structures, or features within the data, making it particularly useful for exploratory data analysis and scenarios where labeled data is scarce or expensive to obtain. This approach includes tasks such as clustering, which groups similar data points, and dimensionality reduction, which transforms data into a lower-dimensional space while retaining its essential characteristics. Unsupervised methods are also crucial in revealing hidden structures and insights in traditional data formats like images or text, and in analyzing the complex relationships present in graph-structured data. Since real-world data rarely comes with ground truth labels, and the generation of labels is time-intensive and usually costly, unsupervised learning plays an important role in exploratory research and knowledge discovery; both are vital in many social and natural sciences.

Graph neural networks as a branch of deep learning in non-Euclidean space have become an important tool in bioinformatics (Zhang et al., 2021). Graphs are for example used to represent protein-protein interaction networks, where nodes represent proteins and edges represent interactions. Unsupervised learning techniques have been used to identify clusters of proteins that work together in biological processes or to predict interactions of proteins that are not known yet, leading to insights into cellular functions and potential drug targets (Gao et al., 2023; Jha et al., 2022; Long et al., 2022).

In the Internet of Things, graph learning is leveraged to analyze and understand the relationships and interactions between connected devices without labeled data. Using a network representation of the Internet of Things, where devices are nodes and their communications or interactions are edges. Unsupervised techniques can uncover hidden patterns or detect unusual behaviors. They are used to enhance security by detecting anomalies in the network (Kim et al., 2022; Wu et al., 2021; Xie et al., 2021). As the Internet of Things networks grow in scale and complexity, graph representation learning approaches can help in managing and analyzing these networks efficiently (Wang et al., 2021).

A classical unsupervised learning task is community detection, which has its roots in social network analysis. The graph structure of a social network is used to identify groups of persons that are more closely connected to each other, than to those outside of their own group. Among many others, the Louvain method is a popular modularity-based approach (also see Section 1.2.1). The detected communities give insight into strong links and social dynamics, they can reveal how information spreads within the network (Wu et al., 2004), show which groups are more influential or how to maximize

influence within a social network (Peng et al., 2018).

There are many techniques for unsupervised machine learning on graphs, such as anomaly detection, link prediction dimensionality reduction, and so on. This thesis focuses on clustering and representation learning. In the following section, we introduce these problems and discuss common methods, applications, and challenges.

1.2.1 Clustering on Graphs

Clustering is one of the most important tools for exploratory data analysis. In many scientific fields that deal with real-world data, clustering is (besides visualization methods) the go-to method. It provides a first impression of the data by highlighting patterns and data points that show similar behavior. On graphs, it is a technique to discover groups of nodes (clusters or communities) that are more densely connected to each other than to the rest of the graph.

More formally, consider a graph G given by its nodes V and links E. The task of clustering is to divide the Graph $G = (V, E)$ into k clusters $C_1, C_2, ..., C_k$ that contain all nodes $V = \bigcup_{i=1}^{k} C_i$ and are disjoint: $C_i \cap C_j = \emptyset$ for all $i \neq j$ with the goal to maximize the connectivity within each cluster C_i while minimizing the inter-cluster connectivity. Depending on the specific problem and the notion of connectivity, there are several possible objectives one can optimize. In the following, we introduce four general approaches to clustering. Note that this list is not exhaustive by any means and merely highlights the diversity of the field.

Methods for graph clustering

Many clustering algorithms are implicitly **modularity-based** (Newman and Girvan, 2004). The core idea is to partition the graph into clusters such that the number of edges within each cluster is significantly higher than would be expected in a random graph with the same degree distribution. In other words, modularity quantifies how well a given graph partition captures the community structure. Popular algorithms based on the modularity objective are the Louvain (Blondel et al., 2008) or the Newman method (Newman, 2006). These algorithms capture precisely this property. However, other algorithms, like spectral clustering, also implicitly optimize a modularity objective.

Spectral clustering, first introduced by Donath and Hoffman (1973) and Fiedler (1973), describes a set of methods that leverage the graph's spectral properties. These methods are based on the eigenvalues and eigenvectors of a matrix representing the graph. This matrix is usually called the graph Laplacian. However, there is no convention on which exact matrix this is. All the same, the eigenvalues of this matrix are used to embed the graph nodes in Euclidean space. In this representation, a conventional clustering algorithm, typically k-means, is used to cluster the nodes. The strength of spectral clustering lies in its ability to handle complex, non-convex clusters, making

it a versatile tool for various types of data and network structures. For a detailed tutorial on spectral clustering, we refer to Von Luxburg (2007). The Laplacian also holds information about the (cluster) structure of the graph; for example, the number of connected components can be derived from the multiplicity of the zero eigenvalues of this matrix and the spectral properties are also linked to random walk behavior on the graph.

Motivated by this connection, there is a set of algorithms that leverage the idea of random walks on a graph to identify dense regions. The idea behind **random walk-based** clustering algorithms is that if one randomly moves from node to node in a graph considering only neighboring nodes, one tends to stay within densely connected regions of a graph for a long time, while one only rarely moves between different clusters. By analyzing (a set) of random walks, it is possible to detect groupings of nodes. The Markov clustering algorithm (MCL) introduced by Dongen (2000) is a fast algorithm for graph clustering that simulates random walks. The algorithm iterates between two operations, expansion, and inflation. The expansion step simulates random walks of many steps and assigns new probabilities to all pairs according to the number of random walks connecting the two. Pairs lying in the same cluster will, generally, be connected by a path more likely than pairs from one to the other cluster. The inflation step boosts the intra-cluster probabilities by taking the power of the probability combined with scaling such that the resulting adjacency matrix is probabilistic (again). Eventually, enough iterations result in the separation of the graph into different segments.

Hierarchical clustering is a technique that builds not a single clustering but, as the name implies, a hierarchy of clusters. We generally distinguish between two main approaches: agglomeration, where clusters are successively merged into larger ones (starting with every node being its own cluster), and divisive, where clusters are successively split into smaller ones until every cluster is a single node.

Besides the definition of clustering above, there is also **soft clustering** which does not aim to find disjoint clusters but allows for overlap in the cluster structure Peters et al. (2013). Instead of assigning a fixed cluster membership, they assign a probability to each node belonging to a cluster. This approach is valuable for two main reasons: firstly, this is, by definition, the better approach if we expect our data to contain overlapping clusters. Secondly, even in the case where we face a clearly separated cluster structure, this approach implicitly returns something we can interpret as a confidence score for a data point to belong to a specific cluster. While there are soft clustering algorithms that are adapted for hierarchical clustering and the other way around, we are not aware of any technique that naturally brings the two perspectives together. In the first part of Chapter 2 we contribute to the clustering literature with an algorithmic framework that inherently combines soft and hierarchical clustering.

Challenges of graph clustering

There are some obstacles all machine learning algorithms have to tackle: Noise and outliers always impose a challenge, and almost all methods face scalability issues when the datasets become very large. One of the challenges that is more present in clustering, as an unsupervised machine learning task, is the evaluation and validation of results. As there is no ground truth to compare to, we need to define independent evaluation metrics to judge their performance. In any case, one needs to be aware of the biases introduced by the respective objective. Another option, which is especially useful in empirical data evaluation, is to use an interpretable method and let experts judge the outcome of the algorithm. This is useful for performance evaluation and generally valuable in real-world explorative research, which focuses on understanding the rationale of the clustering output. While there exists a set of post-hoc explanation algorithms, it is generally unclear whether post-hoc explanations capture the true decision rules of the algorithm. Interpretable clustering algorithms exist, but most of them, like decision trees and other rule-based algorithms, are commonly outperformed by other state-of-the-art algorithms and fail to capture complex, highly non-convex data. In the first part of Chapter 2, we introduce and evaluate a clustering algorithm that, in many cases, allows for interpretable results.

1.2.2 (Unsupervised) Graph Representation Learning

Let us consider a homophile graph, which is a graph where links represent similarity between the nodes. Graph representation algorithms, also called graph embedding algorithms, embed the affinity graph into a new space, which usually has much lower dimensionality. This is especially useful when the nodes are very high dimensional. When choosing the embedding two or three-dimensional, this enables easy visualization of the graph, which is a very powerful tool also for lower dimensional graphs. The general idea is to find a low-dimensional manifold structure hidden in the high-dimensional data geometry reflected by the graph so that the distance in the new embedding space reflects the connectivity of nodes in the graph. For a more elaborate introduction to graph representation learning, we refer to Hamilton (2020) and Zhang et al. (2018).

Many graph representation algorithms can be cast in an encoder-decoder framework, where the input and output of the algorithm are a version of the graph, and the encoding is the low-dimensional embedding of the graph. From this perspective, the algorithm consists of two key components. The encoder is a function that maps each node in the graph to a low-dimensional representation. The role of the decoder is to reconstruct the structural properties of the graph from this low-dimensional embedding. While many decoders are possible, most of them are defined as pairwise decoders, taking pairs of nodes and deciding whether they are connected by a link or not. The straight forward choice for a deterministic decoder is any kind of distance measure together with a simple threshold; every pair of two nodes that are closer together than the specific

threshold are connected by a link. The most commonly used one is the inner product as a distance measure scaled through the sigmoid function to be close to 0 or 1. Close to 1 would indicate a link, and close to 0 would indicate no link.

The encoder and decoder are usually trained end-to-end. For a deterministic decoder, one can often collapse the decoder structure into the loss function. This does not change anything but gives rise to another interpretation of the setup: We only train the encoder and have a sophisticated loss to measure the performance. In this way, the focus generally lies on the encoder. By fixing the decoder function to a graspable function, like the inner product, we gain more control over the inductive biases introduced by this part of the algorithm.

For the sake of completeness, we would like to mention that there are supervised graph representation methods (Li et al., 2020). If one replaces the decoder with, for example, a classifier and chooses some classification loss as the objective, the encoder part can be trained end-to-end in a supervised manner. In this way, any supervised graph neural network task can be interpreted as a supervised representation learning task.

In the following, we briefly discuss the most common methods for graph representation learning.

Methods of graph representation learning

Let us consider a graph given by some diffusion matrix S; this could be the adjacency matrix, the graph Laplacian, or any matrix that captures the desired connectivity structure of the graph. Predicting the neighborhood structure of a node is closely related to predicting the adjacency matrix of the graph. So, one way to think about the encoder-decoder setting is via matrix factorization. The encoder embeds every node of the graph as a low-dimensional vector z. Let Z be the matrix containing all node representations as rows. These methods optimize the distance between the diffusion matrix and the inner product of the node representations: $\mathcal{L} = ||ZZ^T - S||_2^2$. In the encoder-decoder framework, we say these methods use an inner product decoder $\hat{S} = ZZ^T$ and optimize the $L2$-loss: $\mathcal{L} = ||\hat{S} - S||_2^2$. The optimal Z is a low dimensional factorization of the target S. This is why these methods are referred to as **matrix-factorization approaches**.

Random walk methods extend the idea of the inner-product methods. The key innovation in these approaches is that node embeddings are not optimized to approximate distances in the adjacency matrix but rather such that two nodes have similar embeddings if they tend to co-occur on short random walks over the graph. Two popular approaches are the `node2vec` (Grover and Leskovec, 2016), and the `deepwalk` algorithm (Perozzi et al., 2014). Both methods also use the inner-product decoder and mostly differ in their definition of node similarity. While matrix-factorization methods try to reconstruct a deterministic function of the adjacency matrix A, random walk methods optimize embeddings to encode the statistics of random walks. As a result,

the decoder is stochastic and asymmetric.

With a more complex decoder structure, optimizing the encoder becomes increasingly difficult, calling for more universal encoder structures. With the emergence of graph neural networks, they have also been introduced to graph representation learning as a flexible method that introduces little bias into the encoder. However, it is a common mistake to confuse models that are difficult to understand or describe theoretically with models that introduce little inductive bias. We shed some light on the inductive bias of graph neural networks in the second part of Section 1.3. Graph neural networks bring some benefits over the methods already mentioned: They allow the inclusion of node features into the embedding, which is a huge advantage for attributed graphs. They allow for parameter sharing, so nodes are treated similarly and not independently. As a result, graph neural networks can be used to embed new, unseen nodes, even after training is finished. We will talk more about graph neural networks for representation learning in Section 1.2.3.

Challenges of graph representation learning

Graph representation learning faces several critical challenges. First, it inherits all the challenges that machine learning on graphs faces in general: The inherent complexity of graphs, which are irregular and non-Euclidean, makes it difficult to apply traditional machine learning techniques. This challenge is heightened in high-dimensional and heterogeneous networks, where preserving both local and global graph properties is crucial. Additionally, scalability is a significant issue, particularly in large, as existing methods struggle to efficiently process the ever-growing number of nodes and edges in real-world graphs. Furthermore, many real-world graphs are incomplete or contain noisy data, complicating the learning of accurate and robust representations. More specific to representation learning is the hurdle to ensure that models generalize effectively across different types of graphs and domains. Many current models are designed with specific types of graphs in mind and often are only optimized on a single graph. Lastly, integrating domain-specific knowledge into graph representation learning while maintaining interpretability poses an ongoing challenge, particularly in fields like biology and social sciences. As often for unsupervised tasks, it remains difficult to derive explanations for the outcome. In addition, when embedding into a low-dimensional space, we usually lose some of the information. Understanding which part of the features we neglect is generally helpful. More specifically, when applying interpretable downstream tasks to the graph embedding or as a simple sanity check before applying any downstream tasks, it is valuable to understand the rules on which the representation is based.

Leveraging graph neural networks for graph representation learning has addressed some of these challenges. In the next section, we will discuss the benefits of graph neural networks for representation learning in detail.

1.2.3 Representation learning with graph neural networks

In the following section, we introduce the general concept of graph neural networks before discussing them in detail for representation learning.

Graph neural networks

Artificial Neural Networks have achieved great accomplishments in many machine learning tasks, and, measured by accuracy, they outperform traditional machine learning methods in various problems. However, they are not well suited for graph data due to the graph's irregular structure, variable size, and unordered nodes. In recent years, different methods have been proposed to apply the idea of artificial neural networks to graph data; most can be summarized under the term graph neural networks, which also proves to be the most successful approach.

The following sections introduce the basic concept, some of the most generic architectures for graph neural networks (see Figure 1.1), and some basic definitions.

Consider a graph G given by its node features $X \in \mathbb{R}^{n \times d}$, where each row $x_i \in \mathbb{R}^d$ corresponds to the features of one node v_i, and a matrix S, which we will call the diffusion matrix. As before S can be any matrix that represents the graph structure, however usually S is the adjacency matrix, some polynomial of the adjacency matrix or a Laplace matrix of the graph. One of the key properties of a graph neural network-layer is permutation equi- and permutation invariance:

Definition 1 (permutation invariance). We call a function $f : \mathbb{R}^{n \times n}, \mathbb{R}^{n \times d} \mapsto \mathbb{R}$ on a graph $G = (S, X)$ permutation invariant if and only if:

$$f(PSP^T, PX) = f(S, X)$$

for all permutation matrices P.

Definition 2 (permutation equivariance). We call a function $f : \mathbb{R}^{n \times n}, \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times g}$ on a graph $G = (S, X)$ permutation equivariant if and only if:

$$f(PSP^T, PX) = Pf(S, X)$$

for all permutation matrices P.

This ensures that regardless of the order in which they are presented to the algorithm, the result for each node and the graph will not be influenced by shuffling.

A graph neural network cosists of several layers, and in each layer each node is updated by some function of the aggregated information of its neighborhood.

Definition 3 (neighbourhood). We call $\mathcal{N}_i = \{j : (i, j) \in E \vee (j, i) \in E\}$ the neighbourhood of node i, where E is the edge set of graph G.

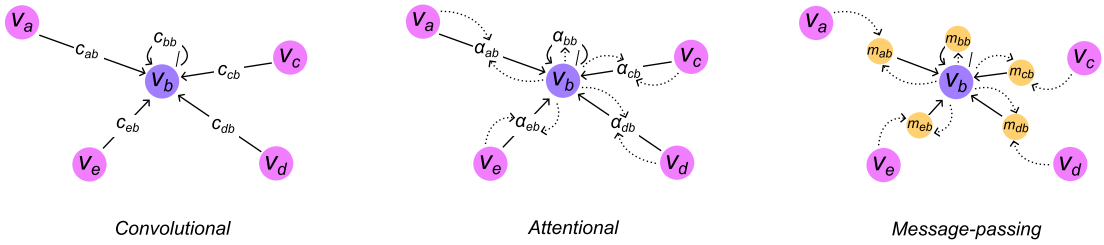


Figure 1.1: Three simple layers-architectures for graph neural networks. The functions are applied locally to each node to update the latent representations.

Figure adapted from Veličković

Definition 4 (featureset). We define the set of features of the neighborhood as: $X_{\mathcal{N}_i} = \{x_j : j \in \mathcal{N}_i\}$

Every layer f_k of the graph neural network is constructed by a shared, locally permutation-invariant function $g_k(x_i, X_{\mathcal{N}_i})$. This function is applied to every node independently before updating the latent representation of the node h_i. The simplest version is called `graph convolution` (Kipf and Welling, 2017) and only aggregates the feature information of the neighborhood with fixed weights c_{ij}. The weights usually depend directly on the adjacency matrix A. For an unweighted graph, they are often chosen to be 1.

Definition 5 (graph convolution). Let ϕ and ψ be (non-linear) functions and $\oplus$ a permutation invariant aggregation function. Then

$$h_b = \phi(x_b, \bigoplus_{j \in \mathcal{N}_b} c_{b,j} \ \psi(x_j))$$

is the local update function of the graph-convolutional layer.

A slightly more complex version is the `graph attention` network Veličković et al. (2018), which adds an attention mechanism in the form of an additional function. This function learns the relevance of each element in the input data to the current task and assigns an attention score. These scores are then used to control the weight between neighboring nodes, where more relevant nodes are emphasized.

Definition 6 (graph attention). Let ϕ and ψ be (non-linear) functions, $\oplus$ a permutation invariant aggregation and α the attention mechanism. Then

$$h_b = \phi(x_b, \bigoplus_{j \in \mathcal{N}_b} \alpha(x_b, x_j) \ \psi(x_j))$$

is the local update function of the graph-attention layer.

The most generic approach is the `message-passing` layer, which computes arbitrary "messages" for all links. While the literature usually differentiates between the methods, one could model the previous architectures with the message-passing approach.

Message-passing was introduced as a framework for general graph neural network layers (Gilmer et al., 2017) rather than a specific architecture.

Definition 7 (message-passing). Let ϕ and ψ be any (non-linear) functions and $\oplus$ a permutation invariant aggregation. Then

$$h_b = \phi(x_b, \bigoplus_{j \in \mathcal{N}_b} \psi(x_b, x_j))$$

it is the local update function of the message-passing layer.

The three introduced layer types capture and aggregate local information. By stacking multiple layers, the latent representations of the nodes can capture information from neighbors farther away in the graph. In every step, the latent representation of each node is updated. Based on this representation, traditional machine learning algorithms can be used to either cluster the nodes or solve a classification or regression task at the node level or the graph level. For the latter, a final output layer is usually used, which is either permutation-equivariant for node-level tasks or permutation-invariant for tasks on the graph level. Such a latent representation can also be interpreted directly as a node embedding, which hints at the practicality of graph neural networks for graph representation learning. Implicitly, any graph neural network that solves a classification or regression task can be interpreted as a two-step procedure, first embedding the graph in latent space and then solving the task. Often, these two steps are trained in an end-to-end manner. However, if we only do graph representation learning, we can focus on the first task only. With the ambition to allow for less bias in the embedding, we can do so in an unsupervised style.

Graph auto-encoder for representation learning

Let us recall the encoder-decode perspective on representation learning discussed in Section 1.2.2. We have an encoder part that takes the graph information given by its features X and a diffusion matrix S which maps the graph to a low dimensional latent space. The decoder part takes this representation and tries to reconstruct the graph information. For (non-probabilistic) graph auto-encoders, as introduced by Kipf and Welling (2016), the diffusion matrix $S = D^{1/2}AD^{1/2}$ is set to be the symmetrically normalized adjacency matrix of the graph with self-loops, that is the diagonal of A is all ones which means all nodes are connected to themselves. The encoder is a simple graph-convolutional network with two layers and a linear output activation function σ:

$$Z = GCN(S, X) = S\ \texttt{relu}(SXW_0)\ W_1$$

The graph-convolution is homophile by design and embeds points that are densely connected closer to each other.

The decoder is deterministic and measures the inner-product similarity of all nodes, thresholded using a sigmoid activation:

$$\hat{A} = \sigma(ZZ^T)$$

This introduces a bias into the embedding space that forces points that are connected in the graph to have a high inner-product similarity, which is dominated by a similar angle.

The loss function that is optimized by the auto-encoder is the cross-entropy between the adjacency matrix A and the reconstructed adjacency matrix $\hat{A}$:

$$\mathcal{L}_A(Z) = \sum_{i,j=1}^{n} a_{ij} \log(\sigma(z_i z_j^T)) + (1 - a_{ij})(1 - \log(\sigma(z_i z_j^T))).$$

Note that this loss depends on the considered graph and is optimized for one graph and thus one embedding only. The nodes are considered the individual data points and this approach can be trained on the whole graph or on individual batches considering only parts of the graph for each gradient decent step.

Observing the modularity of the approach, any part of the pipeline could be replaced arbitrarily, introducing different biases into the architecture. A lot of work extends this generic approach to build more sophisticated models: Ahn and Kim (2021); Davidson et al. (2018); Li et al. (2022); Weng et al. (2020); Xiao et al. (2023) to name a few.

1.2.4 Representation learning for graph clustering

Sometimes, the complexity of graphs is so intricate that we might want to work with a simpler type of data. Representation learning is a popular approach that combines dimensionality reduction with embedding into a different space. In the following, we discuss the main motivations for working with a low-dimensional representation of a graph instead of the original graph. While there are many tasks one can solve on a graph, for this thesis, we focus on clustering.

Raw graphs are often high-dimensional in their nodes and sparse in their edges. Both these challenges often even increase when dealing with larger networks. Some tasks, such as clustering, on such data tend to be computationally expensive and sensitive to noise. Learning a low-dimensional representation of the graph can reduce the complexity and noise, as for the dimensionality reduction it is forced to only capture essential structural feature information. This makes a task like clustering more efficient and robust. In turn, the increase in efficiency also makes clustering more scalable to larger networks.

Some direct clustering methods, such as spectral clustering, consider only the graph

structure, that is the connectivity pattern usually given by the adjacency matrix. This might neglect patterns represented in the node features of the graph. Most representation learning techniques include more complex dependencies and learn embeddings that reflect both, features and graph structure. This might help clustering performance for complex graph data.

In general, including representation learning as a possible first step leads to a larger variety of clustering algorithms as not only graph clustering algorithms are applicable. This allows for experimentation with different approaches.

As a side effect, graph embeddings are not limited to one task and, once computed, can be reused for various downstream tasks, not just clustering. This makes the approach versatile.

In summary, representation learning enables the transformation of complex, high-dimensional, and noisy graph data into a more manageable and structured form. This transformation allows for more effective, robust, and flexible clustering, often leading to better performance, robustness and scalability compared to direct clustering methods.

1.3 Scope and Contribution

This thesis is based on the following publications.

S. Klepper, C. Elbracht, D. Fioravanti, J. Kneip, L. Rendsburg, M. Teegen, and U. Von Luxburg. Clustering with tangles: Algorithmic framework and theoretical guarantees. *Journal of Machine Learning Research (JMLR)*, 24(190):1–56, 2023

In this work we present a novel clustering approach based on the concept of tangles, a fascinating tool from mathematical graph theory. This concept is used to identify and analyze densely connected subsets within a graph. Our work extends this concept to clustering problems, offering a new algorithmic framework and providing theoretical guarantees to support its effectiveness. In graph theory, tangles are regions within a graph where nodes are densely connected. We adapt this idea to the clustering context, where the goal is to identify subsets of data points exhibiting strong internal connectivity relative to their external connections.

The proposed algorithmic framework revolves around identifying tangles in a dataset. We begin by defining what constitutes a tangle in this context and then develop an algorithm to find these tangles. Once identified, these tangles are used to extract clusters. For example, consider a set V of n people answering m binary questions. The aim is to group individuals with similar mindsets. Each question creates a cut in the set V, separating individuals based on their answers (e.g., "yes" vs. "no"). This process generates a collection of m cuts, which we evaluate using a cost function c to

assess their informativeness. If possible, this cost function should incorporate domain-specific knowledge. The cuts are processed in increasing order of their cost, allowing the discovery of a hierarchy of substructures, starting from broad distinctions (e.g., extroverts vs. introverts) to finer distinctions as more detailed cuts are considered. The resulting clustering hierarchy is of a soft nature; the algorithm does not directly assign points to a cluster but instead derives a probability of membership to each respective cluster. This approach allows for the exploration of cluster structures within a dataset, resulting in what we refer to as a "soft dendrogram".

We provide theoretical guarantees for the stability and consistency of the clusters identified across different data domains, including graphs, questionnaire data (binary features), and metric datasets. The performance of the algorithm is evaluated both theoretically and empirically, demonstrating its efficiency and practical feasibility.

The primary contribution of the paper is its innovative approach to clustering by leveraging the concept of tangles. The approach is particularly useful for clustering complex datasets, where traditional methods may fail, such as those with overlapping or intricately structured clusters. If the cuts are interpretable, the tangles corresponding to each cluster provide an explanation of the cluster which is very powerful for analyzing complex datasets. This paper contributes to both the theoretical and practical aspects of clustering, offering new insights and a new tool for tackling challenging clustering problems.

S. Klepper and U. von Luxburg. Relating graph auto-encoders to linear models. *Transactions on Machine Learning Research (TMLR)*, 2023

In this work we investigates the representational power of graph auto-encoders (GAEs) compared to linear models and contribute to understanding the underlying inductive bias of the graph auto-encoder model.

Many real-world problems have a natural representation in terms of a graph that illustrates relationships or interactions between entities (nodes) within the data. Extracting and summarizing information from these graphs is the key problem in graph learning. Graph auto-encoders are popular for encoding graph-structured data in Euclidean space and are widely used in tasks such as link prediction and node classification. However, recent findings show that linear models often outperform these complex non-linear models, which raises the question of the actual necessity of non-linearities in graph auto-encoders. To address this, we analyze the solution space of GAEs and its relation to linear embedding models.

We theoretically demonstrate that the solution space of graph auto-encoders is a subset of that of a linear map, implying that linear models have at least the same representational power as graph auto-encoders that rely on graph convolutional networks (GCNs). This observation suggests that the relevant inductive bias of GAEs may not come from

their non-linearity but rather from the use of node features that enhance learning and generalization. We observe that restricting the linear solution space with these features can also lead to improved model performance.

We provide empirical evidence to support these claims, showing that linear encoders can outperform non-linear encoders when feature information is properly utilized. More specifically we show that for any function $f(G)$ on a fixed graph $G = (A, X)$ there exists a linear model that can achieve the same training loss as f. This result suggests that the non-linearities in GCN-based auto-encoders may not be as crucial as previously believed, shifting the focus towards the role of node features in graph learning. We present theoretical insights by introducing a corresponding bias in linear models and analyzing how it alters the solution space. Our findings indicate that the success of graph auto-encoders is more likely due to the structure of node features rather than the non-linearity of the model itself.

We conclude that the commonly held belief in the superiority of non-linear graph auto-encoders should be reconsidered. Linear models, when designed with the appropriate inductive biases, can achieve competitive results in graph representation learning tasks. With our research, we aim to encourage the community to pay closer attention to the inductive biases, in this case, the role of node features in graph learning, rather than automatically favoring more complex models. Doing so, we provide a fresh perspective on the effective use of linear models in graph-based learning.

Additional Contributions

During my PhD, I contributed to another publication as a co-author.

K. E. Burger, S. Klepper, U. von Luxburg, and F. Baumdicker. Inferring ancestry with the hierarchical soft clustering approach tanglegen. *Genome Research*, 2024

Our publication presents tangleGen, a hierarchical soft clustering tool based on the Tangles framework. It adapts the basic approach of Tangles to infer population ancestries by incorporating population genetics concepts like the fixation index and Hardy-Weinberg equilibrium into the cost function. tangleGen combines numerous local bi-partitions derived from genetic data to form an expressive clustering and prioritizes those with lower cost, that is, higher discriminative power. Tested on both simulated and real data, tangleGen performs comparably to existing methods but stands out for its interpretability and robustness. Its hierarchical structure allows automatic identification of related populations without requiring the number of populations to be pre-specified. The method yields deterministic results across different population sizes and is highly customizable, making it a versatile tool for investigating genetic diversity and ancestral relationships in populations.

In my last year, I also worked on diffusion models.

S. Klepper. Towards understanding diffusion models (on graphs). *arXiv:2409.00374*, 2024

While my primary goal was to understand these models within the context of graphs, I initially conducted experiments in a more straightforward setting to build foundational insights. By empirically investigating various diffusion and sampling techniques, I address three key questions: (1) What is the impact of noise in these models? (2) How does the choice of sampling method influence the results? (3) What function is the neural network approximating, and is high complexity necessary for optimal performance? Our findings seek to deepen the understanding of diffusion models and, ultimately, their application in graph-based machine learning.

Chapter 2

Contributions

Journal of Machine Learning Research 24 (2023) 1-56 Submitted 9/21; Revised 12/22; Published 5/23

Clustering with Tangles: Algorithmic Framework and Theoretical Guarantees

Solveig Klepper[1] SOLVEIG.KLEPPER@UNI-TUEBINGEN.DE
Christian Elbracht[2] CHRISTIAN.ELBRACHT@UNI-HAMBURG.DE
Diego Fioravanti [1] FIORAVANTI.DIEGO@GMAIL.COM
Jakob Kneip[2] JAKOBFKNEIP+JMLR@GMAIL.COM
Luca Rendsburg[1] LUCA.RENDSBURG@UNI-TUEBINGEN.DE
Maximilian Teegen[2] MAXIMILIAN.TEEGEN@UNI-HAMBURG.DE
Ulrike von Luxburg[1] ULRIKE.LUXBURG@UNI-TUEBINGEN.DE

[1] *Department of Computer Science and Tübingen AI Center, University of Tübingen, Germany*

[2] *Department of Mathematics University of Hamburg Germany*

Editor: Samory Kpotufe

Abstract

Originally, tangles were invented as an abstract tool in mathematical graph theory to prove the famous graph minor theorem. In this paper, we showcase the practical potential of tangles in machine learning applications. Given a collection of cuts of any dataset, tangles aggregate these cuts to point in the direction of a dense structure. As a result, a cluster is softly characterized by a set of consistent pointers. This highly flexible approach can solve clustering problems in various setups, ranging from questionnaires over community detection in graphs to clustering points in metric spaces. The output of our proposed framework is hierarchical and induces the notion of a soft dendrogram, which can help explore the cluster structure of a dataset. The computational complexity of aggregating the cuts is linear in the number of data points. Thus the bottleneck of the tangle approach is to generate the cuts, for which simple and fast algorithms form a sufficient basis. In our paper we construct the algorithmic framework for clustering with tangles, prove theoretical guarantees in various settings, and provide extensive simulations and use cases. Python code is available on github.

Keywords: tangles, clustering framework, soft clustering, hierarchical clustering, interpretable clustering

1. Introduction

In this paper, we present tangles, a new tool that can be used for clustering, to the machine learning community. Tangles are an established concept in mathematical graph theory. They were initially introduced by Robertson and Seymour (1991) as a mechanism to study highly cohesive structures in graphs and have since become a standard tool in the analysis of other discrete structures (Diestel, 2018). Recently, Diestel (2019) suggested applying the

abstract notion of tangles beyond their original context to data clustering problems. The purpose of our paper is to make this suggestion come true. We translate abstract mathematical notions into practical algorithms, prove theoretical guarantees for the performance of these algorithms, and demonstrate the usefulness and flexibility of the new approach in diverse applications.

The mechanism of tangles is very different from all of the current clustering algorithms we know. To introduce this concept, we consider the example of a personality traits questionnaire, in which a group of persons answers a set of binary questions. Based on the answers, we would like to identify groups of like-minded persons and characterize their associated mindsets, such as being "narcissistic". One would expect that persons sharing a mindset agree on many relevant statements; for example, most narcissists would agree on the statement "I have a strong will to power". Accordingly, we would like to *softly characterize* a mindset by saying that most persons with this mindset answer similarly to most questions. We can formalize this idea using tangles. First, we interpret every question as a bipartition of all the persons who participated in the questionnaire. This bipartition (equivalently, cut) splits the set of persons into the ones answering "yes" versus the ones answering "no". Let us assume that most persons who share a mindset give the same answers to most questions. Visualized in terms of cuts, we can say that persons of the same mindset tend to lie "on the same side" of most of the cuts. We now assign an orientation to each of the cuts to identify one side of the respective bipartition: we orient the cut to "point towards" the group of persons. Assume for the moment that we already know the mindset that we want to describe. The description then consists of the chosen orientations, indicating the "typical way" of answering all the questions. Conversely, the orientations of all the cuts identify a group of persons: the persons that the cuts point towards. See Figure 1.

More generally, the tangle framework is as follows. Given a dataset, in the first step, we construct a set of bipartitions of the data. These cuts can be constructed in a quick and dirty manner; all we need is that they provide a little information regarding the cluster structure of the points. In a second step, we then find "consistent" orientations of these cuts. Typically, there will be several consistent orientations. Each of them is one particular "tangle" of the data. In a final step, the tangles can then be converted into meaningful output, for example, a hard or soft clustering of the dataset or even a soft dendrogram (see Figure 2).

What are the benefits of this approach? The tangle approach is very general and highly flexible. Instead of assigning cluster memberships to individual objects, tangles characterize a cluster indirectly by a set of pointers. This flexible representation mitigates the problem of dealing with ambiguous cases and naturally entails a hierarchical structure. Tangles require as input only a collection of cuts of the dataset. When choosing these cuts, we can incorporate prior knowledge that we might have about our problem. We do not require a particular data representation. Quite the contrary: tangles can be applied to many different scenarios such as feature-based data, metric data, graph data, and questionnaire data. For exemplary use cases see Sections 4, Section 5 and Section 6. From a conceptual point of view, tangles resemble the boosting approach for classification, where one aggregates many weak classifiers – slightly better than chance – to obtain a strong classifier.

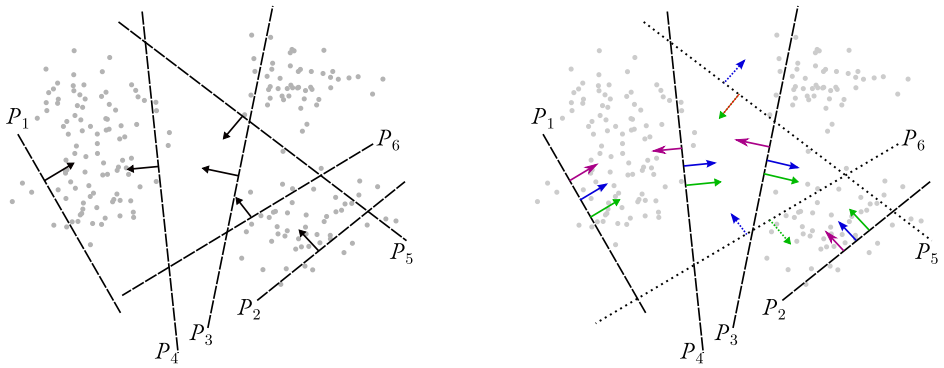


Figure 1: We consider a set of points and six cuts. The left image visualizes one possible tangle (consistent orientation). The right image visualizes three additional tangles, that exist on (sub)sets of the same cuts. When constructing the tangle search tree, we would first obtain two tangles on the set $\{P_1, ..., P_4\}$: the purple and the green/blue tangle. The green and the blue tangle share the orientations of the more informative cuts but differ in bipartition P_5 and P_6, indicated by dashed arrows. Lower down in the hierarchy, we get three tangles on the whole set of cuts $\{P_1, ..., P_6\}$: green, blue and the black tangle visualized in the left picture.

Tangles aggregate many "weak" cuts that contain a large chunk of a cluster on one side to obtain a holistic, "strong" view of the cluster structure of a dataset. The computational complexity of the tangle approach is composed of two parts: constructing cuts in the pre-processing phase and orienting the cuts in the central part of the algorithm. This central part of the algorithm is only linear in the number of data points. That means that given a simple way of constructing a set of cuts in the pre-processing phase, the whole approach is fast and works for large-scale datasets.

Our contributions are as follows:

- **Algorithmic framework.** We translate the abstract notion of tangles from the mathematical literature to a more practical version for machine learning in Section 2. We then develop a highly flexible algorithmic framework for clustering. We propose a basic version in Section 3 and refer to Appendix II for further extensions and details.

- **Simulations and experiments.** To demonstrate the flexibility of the tangle approach, we provide case studies in three different scenarios: a questionnaire scenario in Section 4, a graph clustering scenario in Section 5, and a feature-based scenario in Section 6. In each of these sections, we outline different properties of the tangle approach. Generally, we compare tangles to other state-of-the-art algorithms in the respective domains, for example, spectral clustering in the graph clustering domain or k-means in the feature-based domain.

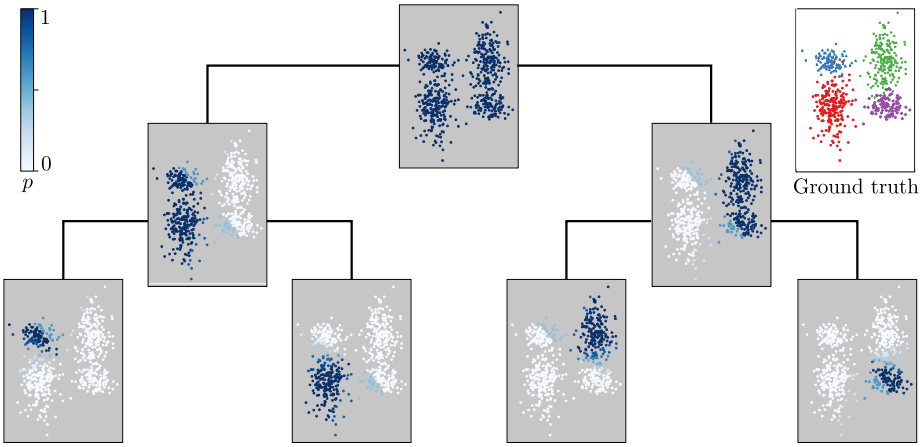


Figure 2: A soft dendrogram as possible post-processing of tangles (Appendix 3.4). The estimated probability that a point belongs to the respective cluster is given by p.

- **Theoretical guarantees.** In each of the three scenarios, we prove theoretical guarantees. Given a statistical model for the questionnaire setting, we prove that tangles always discover the ground truth under specific parameter choices. We prove the same for the graph clustering scenario in a stochastic block model. Finally, we investigate theoretical guarantees on feature-based clustering for interpretable clustering.
- **Python package.** We implemented the central part of the algorithm and different options for pre- and post-processing. The code and basic examples are publicly available at: https://github.com/tml-tuebingen/tangles/tree/vanilla.

The strength of tangles is not that they outperform all other algorithms; this would be pretty unrealistic. Instead, we are intrigued by how flexible and how generic the tangle approach turns out to be, while at the same time producing results that are comparable to many state-of-the-art algorithms in many domains. All in all, we consider this paper as a proof of concept for a completely new approach to data clustering.

2. Tangles: Notation and definitions

Tangles originate in mathematical graph theory, where they are treated in much more generality than what we need in our paper (cf. Diestel (2018) for an overview, and Section 7 for more discussion and pointers to literature). Through our joint effort between mathematicians and machine learners, we condensed the general tangle theory to what we believe is the essence of tangles needed for applications in machine learning. We present this condensed version below. For readers with a mathematical tangle theory background, we provide a translation dictionary of the essential terms in Appendix I.1.

Consider a set $V = \{v_1 \dots, v_n\}$ of arbitrary objects. A subset $A \subset V$ induces a **bipartition** or **cut** of the data into the set and its complement $P = \{A, A^{\complement}\}$. In order to construct tangles, we will consider a set of initial cuts $\mathcal{P} = \{\{A_1, A_1^{\complement}\}, \dots, \{A_m, A_m^{\complement}\}\}$. We consider

a single cut useful if it does not separate many similar objects. The more it cuts through dense regions, the less insight we get into the cluster structure. This intuition is being quantified in terms of a **cost function** $c : \mathcal{P} \to \mathbb{R}$, indicating the "quality" of a cut. This cost function needs to be chosen application-dependent; see later for examples. The set of bipartitions and associated costs hold all the necessary information for tangles to discover the dataset's structure. Tangles operate by assigning an **orientation** to all cuts. For a single cut $P = \{A, A^{\complement}\}$, an orientation simply "points" towards one of the sides. We denote the orientation pointing from A to $A^{\complement}$ by $\vec{P} = (A, A^{\complement})$ or simply $\vec{P} = A^{\complement}$. For a set of cuts $\mathcal{P}$, we define an orientation $O_{\mathcal{P}}$ by choosing one side for each cut, giving an orientation to every $P \in \mathcal{P}$. We write $A \in O_{\mathcal{P}}$ if $\{A, A^{\complement}\} \in \mathcal{P}$ and $O_{\mathcal{P}}$ orients it towards A. The intuition is that orientations can characterize clusters, but not every orientation of cuts characterizes a cluster: the orientations need to be "consistent" in some way. For a meaningful orientation, we have to ensure that the chosen sides of all the cuts point to one single structure. This consistency is precisely the purpose of tangles and is captured in the following definition.

Definition 1 (Consistency and Tangles) *Let $\mathcal{P}$ be a set of bipartitions on a set V. For a fixed parameter $a \in \mathbb{N}$, an orientation $O_{\mathcal{P}}$ of $\mathcal{P}$ is consistent if all sets of three of oriented cuts have at least a objects in common:*

$$\forall A, B, C \in O_{\mathcal{P}} : \quad |A \cap B \cap C| \geqslant a \,. \tag{1}$$

We call Eq. (1) *the* consistency condition *and a the* agreement parameter*. A consistent orientation of $\mathcal{P}$ is called a* $\mathcal{P}$-tangle*. If clear from the context, we drop the dependency on $\mathcal{P}$ and say* tangle*.*

Note that neither the definition of an orientation nor the definition of a tangle is symmetric. Choosing the other set of each bipartition, i.e. the inverted orientation will point to a different set of points and will not correspond to the same structure in the data.

At this point, the reader might wonder why we consider an intersection of exactly three cuts in Eq. (1). The short answer is that there are good mathematical reasons for this choice. One can prove that considering the intersection of at least three cuts guarantees that there exist at most as many distinct tangles as there are data points — which makes perfect sense in the application of data clustering. If one uses the intersection of only two cuts in Eq. (1), then there might be up to $2^{2|V|}$ many tangles, which is undesirable both from a conceptual as well as a computational point of view. On the other hand, it turns out that choosing sets of more than three in Eq. (1) does not produce more powerful mathematical results, but considerably increases the computational complexity. We explain more details about the question of three in Appendix I.2.

In what follows, we will often sort the cuts according to their cost $c(P)$, and start orienting the (more useful) low-cost cuts before moving on to the (less useful) high-cost cuts. To this end, we sometimes introduce a parameter $\Psi \in \mathbb{R}$ that specifies the set of cuts we are interested in, namely the subset $\mathcal{P}_\Psi = \{P \in \mathcal{P} \mid c(P) \leqslant \Psi\} \subseteq \mathcal{P}$. We say a tangle on the set $\mathcal{P}_\Psi$ is of order Ψ.

In the following, we build intuition on the above definitions using the running example of the questionnaire that we already hinted in the introduction.

Example (Questionnaire) A set V of n persons takes a questionnaire of m binary questions. The goal is to discover groups of persons who answer most questions similarly. If such a group exists, we say they share the same mindset. We interpret each question as a cut in V, separating persons based on their answer to this question. In this way, the questionnaire defines a set $\mathcal{P}$ of m cuts. Generally, the cuts in $\mathcal{P}$ can split V at different levels of granularity, depending on how general a question is. Some of the cuts might not be informative; for example, a person's hair color is mostly independent of other personality traits. To judge on an abstract level how useful a cut might be, we introduce a cost function $c\colon \mathcal{P} \to \mathbb{R}$. In our example, we judge the similarity of two persons, or rather their answered questionnaires $v, w \in \{0,1\}^m$, by counting how many questions they have answered the same way: $sim(v,w) = \sum_{i=1}^{m} \mathbb{1}\{v_i = w_i\}$. We then define the cost of a cut as the mean over the similarities over all possible pairs of separated persons: $c(A) = \frac{1}{|A|\cdot(n-|A|)} \sum_{v\in A, w\in A^\complement} \mathrm{sim}(v,w)$. We now process the cuts in increasing order of costs: most useful cuts come first, and less useful cuts come later. This approach is equivalent to repeatedly setting the threshold Ψ and restricting our attention to the set $\mathcal{P}_\Psi$. Increasing the order Ψ enables us to discover a hierarchy of substructures. For a small order, we can only distinguish between coarse structures (such as extroverts and introverts), while for a larger order, we include cuts that further separate them into more fine-grained structures. For any given Ψ, we need to find an orientation of the cuts in $\mathcal{P}_\Psi$ that "points towards a cluster", as formalized in Definition 1. Concretely, we need to set the agreement parameter a and invoke an algorithm that discovers consistent orientations of all orders of cuts. Once we have found consistent orientations, we need to post-process them to the final output. This output could consist of a description of all mindsets in terms of the typical way of answering questions; it could be a hard clustering of the persons or a soft hierarchical clustering. We will introduce the algorithm and all these notions in the next section.

3. Basic algorithms for tangle clustering

In this section, we present the basic algorithmic framework for clustering with tangles. On a high level, this requires the following three independent steps: finding the initial set of cuts (Section 3.1), orienting cuts to identify tangles (Section 3.3), and post-processing tangles to clusterings (Section 3.4). To allow for a deeper understanding of the framework we give intuition on parameters and how they interact in Sections 3.2 and 3.5. For more algorithmic details we refer to Appendix II. In Sections 4 – 6 we will then spell out all details in three different application settings. Python code of the basic version as well as examples can be found on github [1].

1. https://github.com/tml-tuebingen/tangles/tree/vanilla

3.1 Constructing the initial set of cuts

The first step for finding tangles is to construct a set of initial cuts $\mathcal{P}$. This construction is very much problem-dependent, and in our pipeline for finding tangles, it has the flavor of a pre-processing step. We can distinguish two principal scenarios that occur in different types of applications:

Predefined cuts. In our running example of a questionnaire, each question induces a natural cut of the data space: the persons who answered "yes" versus those who answered "no". The set of the cuts induced by all questions is a natural candidate for the desired set $\mathcal{P}$. In this case, we can interpret tangles as a typical way of answering the questionnaire. More generally, if the objects in V are described by discrete, continuous, or ordinal features, we can consider a collection of half-spaces of the form $\{x_i \leqslant k\}$. The resulting cuts (and consequently, the tangles) have a simple form and result in interpretable output. See Section 4.1 for an example of interpretable clustering.

Cuts by simple pre-processing. If no natural choice for cuts exists or if they are not flexible enough, it is necessary to invoke another algorithm that produces the initial cuts in a pre-processing phase. In this case, we can view tangles as a boosting mechanism that allows us to use a fast, greedy heuristic for producing decent cuts, which then get aggregated to a tangle and can be processed to clustering. One example of such a setting is graph clustering. Here we could construct initial cuts by the Kernighan-Lin (KL) algorithm (Kernighan and Lin, 1970) and then use tangles to infer the cluster structure on the graph. The complexity of this approach is $\mathcal{O}(rn^2 log\ n)$, for n nodes and r iterations of the KL-Algorithm. Another example is clustering in Euclidean spaces, where we can quickly construct initial partitions with the help of random projections in a one-dimensional subspace. The complexity of this approach is $\mathcal{O}(n^2)$.

Below, we will study cut-finding strategies in three specific settings: binary questionnaires (Section 4), graphs (Section 5) and metric/feature data (Section 6). In Section 5.2 and 6.2 we additionally review the influence and trade-off between a large versus a small set of initial cuts, and in Appendix III.2.1 we discuss why purely random initial cuts are not a good idea.

3.2 Setting the key parameters

Once we have fixed a set of partitions, we need to find consistent orientations of these partitions, that is, the tangles. This first requires some parameter choices: we need to define a cost function c of the cuts (to be able to order the cuts according to their usefulness) and choose the agreement parameter a (which is related to the size and the number of clusters we expect to find). A natural choice for the cost function is the sum of similarities between separated objects $c(\{A, A^{\complement}\}) = \sum_{v \in A, w \in A^{\complement}} \mathrm{sim}(v, w)$. We often also normalize this cost function by dividing it by the number of pairs $|A|\,(n - |A|)$. We discuss the influence of normalizing in Appendix II.2.2. The agreement parameter a roughly fixes the smallest size of the clusters that tangles discover. See Section 3.5 for a discussion of all parameters.

3.3 Orienting cuts to identify tangles

Once we have the data and fixed all parameters, we face the following algorithmic challenge:

Given a set of initial cuts $\mathcal{P}$ of V and a cost function, for every Ψ identify all orientations of $\mathcal{P}_\Psi$ that satisfy the consistency condition Eq. (1).

The naive approach of testing every possible orientation for consistency is infeasible. Instead, we are now going to construct a tree-based search algorithm that achieves this task more efficiently. The algorithm proceeds by looking at one cut after the other, starting with the lowest cost cuts. It maintains a tree, the *tangle search tree*, of the possible orientations of all the cuts considered. The critical observation is that processing a tree branch can be stopped once a cut cannot be oriented consistently any more.

The algorithm's output is a labeled binary tree as depicted in Figure 3b. Each node in the tree corresponds to one specific orientation of one particular cut. We construct the tree in such a way that each of its nodes corresponds to exactly one tangle. Precisely, for a node t on level i the node labels on the path from the root to t form a consistent orientation of $\{P_1, \dots, P_i\}$, that is, a $\mathcal{P}_\Psi$-tangle for $\Psi = c(P_i)$.

The tangle search tree algorithm proceeds as follows. We first sort all cuts in $\mathcal{P}$ by increasing cost and list them as $P_1 = \{A_1, A_1^\complement\}, \dots, P_m = \{A_m, A_m^\complement\}$. We now perform something like a breadth-first search on possible orientations. We initialize the tree with an unlabeled root on level 0. We now iterate over the P_i. In the i-th step, for both sides $A \in \{A_i, A_i^\complement\}$ and every node t on level $i-1$, we check whether adding the orientation A to the tangle identified with node t is consistent. If it is, we add a child node to t, labeled with A (see Algorithm 1 and Appendix II.1). In the resulting tree, each node represents a tangle, and each leaf represents a maximal tangle, one that cannot be extended to a tangle of a larger set $\mathcal{P}_\Psi$.

The algorithm has complexity $O(n\ell h^3)$ where n is the number of objects in our dataset, h is the height of the tangle search tree, and ℓ is the number of its leaf nodes. The number of leaf nodes is bound by the number of nodes or the height of the tree: $\ell \leqslant n$ and $\ell \leqslant 2^h$; usually we observe $\ell \ll n$. In practice, we find that the worst-case complexity is rarely attained (Figure 15). The height h is upper bounded by the number of cuts m. The tangles at the leaf nodes correspond to the smallest clusters. The agreement parameter a indirectly controls both ℓ and h. Increasing a makes the Eq. (1) more restrictive and thus cuts the tree quicker.

3.4 Post-processing the tangles into soft or hard clusterings

The output of Algorithm 1 is a tangle search tree, which reveals the cluster structure of a dataset from the cut point of view. Strictly speaking, it is inappropriate to think of tangles as subsets; instead, they "point towards a region" without making statements about individual objects. Nevertheless, traditional clustering objectives are concerned with assigning individual objects to clusters. In order to achieve this with tangles, we post-process the tangle search tree in different ways resulting in hierarchical, soft, and hard

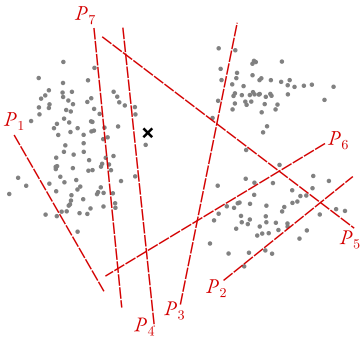
Algorithm 1: tangle search tree

Data: Set of cuts $\mathcal{P} = \{P_i = \{A_i, A_i^{\complement}\}\}_i$ with cost function $c : \mathcal{P} \to \mathbb{R}$, agreement parameter a

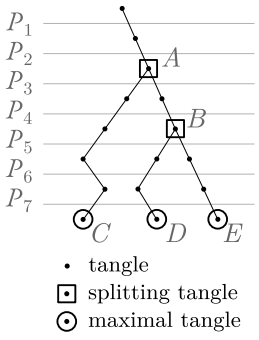
Result: Tangle Search Tree T

```
T ← empty tree with root;
sort P_i increasing according to Ψ_i = c(P_i);
for P_i ∈ P do
    for tangle τ ∈ nodes of layer i of T do
        if consistent( τ ∪ {A_i} ) then
            add A_i as right child of τ to T ;
        end
        if consistent( τ ∪ {A_i^C} ) then
            add A_i^C as left child of τ to T ;
        end
    end
end
return T
```

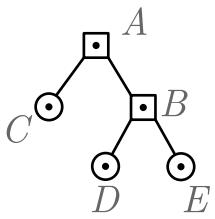
clustering. To this end, we propose a procedure that builds on the hierarchical nature of the tangle search tree to convert it into a "soft dendrogram".



(a) Dataset V and cuts $\mathcal{P} = \{P_1, \ldots, P_7\}$.



(b) Tangle search tree $T(\mathcal{P})$.



(c) Condensed tangle search tree $T^*(\mathcal{P})$.

Figure 3: Example of tangles for a dataset in $\mathbb{R}^2$.

For a given set of partitions $\mathcal{P} = \{P_1, \ldots, P_m\}$ sorted increasingly by their costs $c(P_i)$, let $T = T(\mathcal{P})$ be the corresponding tangle search tree obtained from Algorithm 1, see Figure 3a and 3b for an example. The tangle search tree is constructed hierarchically on the cuts, which serves as a proxy for what we are eventually interested in, namely, a hierarchy of the cluster structure of the objects. As a further simplification, we will explain below how to transform the tangle search tree into a simplified, condensed tree. Like a dendrogram, the condensed tree T^* indicates how a dataset organizes into substructures. We call every internal node a splitting tangle as its two subtrees correspond to tangles that point to dif-

ferent regions and thus split the data. However, for a single object, a splitting tangle does not induce a binary decision as to whether the object belongs to the left or right branch. Instead, we will assign a probability for belonging to a specific tangle for every node and tangle.

Contracting the tree. We first condense the tree to the splitting tangles and ignore bipartitions that do not give information about the cluster structure, for example, P_1. For every splitting tangle, we identify the cuts responsible for the split and thus 'characterizing' for separating the two dense structures. The intuition becomes clear from Figure 3a: For the first splitting tangle τ at the node A, $\tau_{|A}$, we see that the set of cuts $\{P_3, P_4, P_7\}$ gives information about the separation between the left and the right structure $\mathcal{P}(\tau_{|A}) = \{P_3, P_4, P_7\}$. For the splitting tangle $\tau_{|B}$, we get $\mathcal{P}(\tau_{|B}) = \{P_5, P_6\}$, separating the upper from the lower structure on the right side.
We derive this information from the tangle search tree as follows. For a cut P to be characterizing for a splitting tangle τ, we require every tangle corresponding to a leaf in one subtree to orient P one way and every tangle corresponding to a leaf in the other subtree to orient P the other way. Considering the splitting tangle at node A, P_7 is characterizing: it is oriented to the left in all paths in the left subtree and to the right in all paths in the right subtree. The same holds for cuts P_3 and P_4. In contrast, the cut P_6 is not characterizing as it is oriented both; to the left and right side within the right subtree. So $P_6 \notin \tau_{[\mathcal{B}]}$. In this sense, the cuts in $\mathcal{P}(\tau)$ are the ones that help in distinguishing between the subtrees of τ. More formally, let $P(\tau)$ be the orientation of P in a tangle τ and let $T_\tau^{(left)}$ be the left subtree and $T_\tau^{(right)}$ be the right subtree of the node at a tangle τ. Then we define the set of characterizing cuts as

$$\mathcal{P}(\tau) := \{P \in \mathcal{P} \mid \forall \text{ leaves } \tau^l \in T_\tau^{(left)}, \text{ leaves } \tau^r \in T_\tau^{(right)} : P(\tau^l) \neq P(\tau^r)\}.$$

Based on this information, we condense the tree as shown in Figure 3c and track the set of characterizing cuts for each of the splitting tangles.

Computing the soft clustering.
We now use these cuts to determine how likely an object $v \in V$ belongs to the right subtree of τ (the left subtree is then implicit, so we focus on the right side). We chose the set such that all cuts in $\mathcal{P}(\tau)$ serve the same purpose of subdividing τ into two substructures. For every point v and every splitting tangle τ, we compute the fraction of characterizing cuts oriented towards the point v by the overall number of characterizing cuts $|\mathcal{P}(\tau)|$. As not all cuts are equally fundamental as measured by their costs $c(P)$, we include a weighting of the cuts with a non-increasing function $h : \mathbb{R} \to \mathbb{R}, P \mapsto e^{-c(P)}$. $\{P \in \mathcal{P}(\tau) \mid v \in P^{(right)}\}$ is

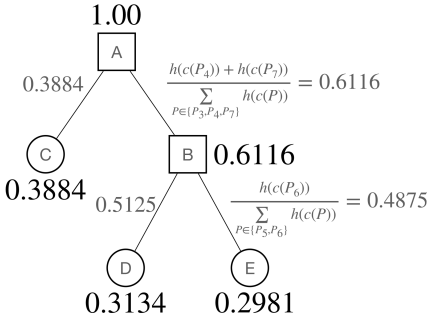


Figure 4: Tree T^* with node and edge attributes for a fixed object $v_x \in V$.

the set of characterizing cuts that are oriented towards v in the right side of the tree. We assign a probability $p_\tau^{(right)}$ of belonging to the right subtree at a tangle τ to every node v:

$$p_\tau^{(\text{right})}(v) = \frac{\sum_{P \in \{P \in \mathcal{P}(\tau) \mid v \in P^{(\text{right})}\}} h(c(P))}{\sum_{P \in \mathcal{P}(\tau)} h(c(P))}. \tag{2}$$

Based on these probabilities, we define the probability $p_t(v)$ that v arrives at node t as the product of the edge probabilities along the unique path from the root to τ. For a single point v_x marked with an x in Figure 3a and the given characterizing cuts we get the tree T^* with the probabilities as shown in Figure 4.

Computing the hard clustering. If desired, we can now assign points to a hard clustering: We assign each point to the tangle with the highest probability based on the soft clustering. For example, the point whose tree is shown in Figure 4 would be assigned to the tangle C represented by the leftmost leaf in the tree. The result is a hard clustering.

In our experiments, we sometimes apply heuristics, such as pruning "bad" branches of the tree, to avoid spurious tangles. We discuss algorithm improvements in Appendix II.2.1.

3.5 The ingredients and how they influence the output

Initial cuts. The utility of tangles depends strongly on the initial set of cuts $\mathcal{P}$ because the tangles' contribution is to aggregate information that is present in $\mathcal{P}$. If there is no cut in $\mathcal{P}$ that separates meaningful substructures, neither will the tangles. The better the cuts, the better the clustering we can derive from tangles. However, choosing the set of cuts is not as critical as one might think for two reasons: (1) Useless cuts do not interfere with tangles: very unbalanced cuts, such as $P = \{\{v\}, V \backslash \{v\}\}$, always get oriented towards their larger side and have little impact on the consistency condition; meaningless cuts, such as random cuts, have high costs and are considered last, quickly resulting in inconsistent orientations only. (2) It is not necessary that $\mathcal{P}$ contains high-quality cuts — otherwise, the whole approach would be somewhat pointless. It is enough to have some "reasonable" initial cuts. We will demonstrate this in experiments and partly in theory below and in the appendix.

The parameter Ψ. The parameter Ψ controls the granularity of the tangles. Restricting our attention to a subset $\mathcal{P}_\Psi$ with a small parameter Ψ will identify large subgroups in the data. As Ψ increases, the corresponding $\mathcal{P}_\Psi$-tangles can identify smaller, less separate clusters, but at the same time, orientations towards larger, more separated clusters may become inconsistent. Eventually, when Ψ gets too large, we might not find any consistent orientation anymore. We typically do not set Ψ to a fixed value but generate a whole hierarchy of clusterings for increasing values of Ψ, as described in Section 3.3.

Agreement parameter a. The agreement parameter a controls the minimal degree to which the sides of an orientation have to agree. When chosen too small, the consistency condition induced by a may be too weak so that tangles identify substructures that we would not consider cohesive. On the other hand, we should not choose a larger than the

smallest cluster we want to discover. Indeed, in practice, a should be slightly smaller than the smallest cluster to allow for noise. The more the cuts in $\mathcal{P}$ respect the cluster structure and especially the richer the set $\mathcal{P}$ is, the more we can reduce a without erroneously identifying incohesive structures as tangles.

4. Use Case: Binary Questionnaire

The most intuitive application for tangles is data coming from a binary questionnaire. In the following, we will give a better intuition about the different aspects of tangles in this practical setting using a simple real-world dataset.

4.1 Case study

As a simple instance, we chose the Narcissistic Personality Inventory questionnaire (Raskin, 1988), sometimes abbreviated npi in the following. Raskin and Hall developed the test in 1979, and it since then has become one of the most widely utilized personality measures for non-clinical levels of the trait narcissism. The dataset is accessible via `https://openpsychometrics.org/_rawdata/` and contains 40 binary questions answered by 11243 participants. Each question consists of a pair of statements, for example, "I am not sure if I would make a good leader" vs. "I see myself as a good leader". See Appendix III.3 for the full list of questions. Every participant is asked to choose the option that they most identify with. If a participant identifies with both equally, they should choose which statement is more important in their opinion. The developers handcrafted an evaluation score for the dataset: For every pair of statements, one statement gets assigned a score of 0, and the other one a score of 1. Each participant's final score s_{npi} is defined as the sum of the scores of the answers, resulting in a number between 0 and 40. The higher the score s_{npi}, the more narcissistic a person is assumed to be. Figure 5 visualizes the frequencies of the participants over the score s_{npi}. We consider s_{npi} as the baseline in the following.
For our experiments, we use each question as a natural bipartition of the persons V into two sets $\{A, A^\complement\}$ where $A \in V$ is the set of persons choosing the first statement. This approach gives us one bipartition for each question, resulting in 40 cuts. To measure the similarity of two participants, we use the Hamming similarity between to answered questionnaires $u, v \in \{0,1\}^m$

$$s(u,v) = \sum_{i=1}^{m} \mathbb{1}\{u_i = v_i\}. \tag{3}$$

To assign a cost to a bipartition $\{A, A^c\}$ we then average this similarity over all pairs of persons of complementary sets:

$$c(\{A, A^\complement\}) = \frac{1}{|A| \cdot |A^\complement|} \sum_{u \in A, v \in A^\complement} s(u,v) \tag{4}$$

From Figure 5, it is evident that this data does not reveal a clear cluster structure when we only consider the score s_{npi}.

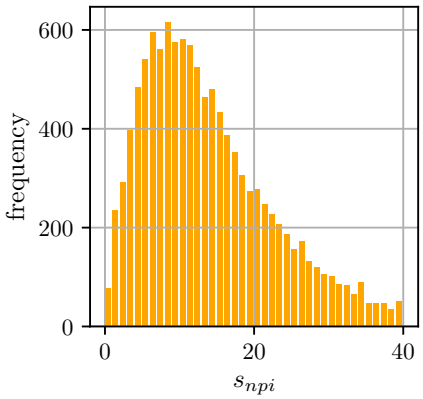


Figure 5: Frequencies of the hand-designed score s_{npi} in the dataset.

Now we study the dataset from the tangle point of view. We naively apply tangles to the whole dataset without pre-processing: we use the data of all 11243 participants, consider all 40 questions as bipartitions and choose a small a of 1500. We use the algorithm described in Section 3.3 to generate the tangle search tree. To avoid clustering on noise, we prune paths of length one, as described in Section 3.4.

The tangle algorithm returns exactly one tangle τ in this setting. This outcome is what we would expect from Figure 5, which already hints that the dataset does not contain a coarse cluster structure. The tangle τ orients 39 of the 40 questions towards one dense structure, and only one question does not get assigned an orientation by τ (question #1). The orientations specified in τ represent the "stereotypical way" by which persons of the corresponding mindset answer all the questions. Recap that this does not necessarily mean that a person in the dataset answered all the questions precisely this way. When we compare these orientations to the hand-crafted orientations by the inventors of the study, we find that τ discovers the "correct" assignments to all 39 questions! This outcome is remarkable: while the original study hand-designed the orientation of the questions (that is, which statement is 1 and which is 0), our algorithm discovers these orientations on its own. The only difference is that τ inverts all orientations: it points toward the larger group of people, which is the group of non-narcissistic persons, while in the original study, the authors oriented the question to point toward the minority group, the narcissistic people. So our first finding is that tangles reveal the same information as the authors hand-crafted into the data, but in a completely unsupervised manner.

We can now try to improve these results. Which questions are most important, and are there questions that we do not need to consider? We run a second experiment to demonstrate how tangles distinguish between different clusters. Based on the discovered tangle τ, we assign a score s_τ to each of the participants: For every participant $u \in V$, we compute the Hamming distance of her answers q_i to the stereotype answers τ_i given by tangle τ: $s_\tau(u) = \sum_{i=1}^{m} \mathbb{1}\{q_i \neq \tau_i\}$. This score measures how much a participant's answers deviate

from the typical non-narcissistic person. s_τ takes values between 0 and 39, and the higher the score, the more narcissistic we believe a person to be. As expected by the fact that the tangle orientation essentially coincides with the hand-crafted orientation, the correlation coefficient between s_{npi} and s_τ is very high, 0.996. We use our new score s_τ to sample a subset of participants that is balanced in terms of the score s_τ: we randomly sample 18 participants that have score $s_\tau = 0$, another 18 participants that have score $s_\tau = 1$, and so forth. This results in a subset of $18 \cdot 40 = 720$ participants.
We now apply tangles to this new dataset. As before, we use all the 40 bipartitions given by the questions and the same cost function as before. We set our agreement parameter a to 150 and prune paths of length one. On this balanced dataset, the tangle algorithm returns two tangles, indicating that within this balanced subset of the data, there is a cluster structure with two dense structures. If we wish to do so, we could now use our hard

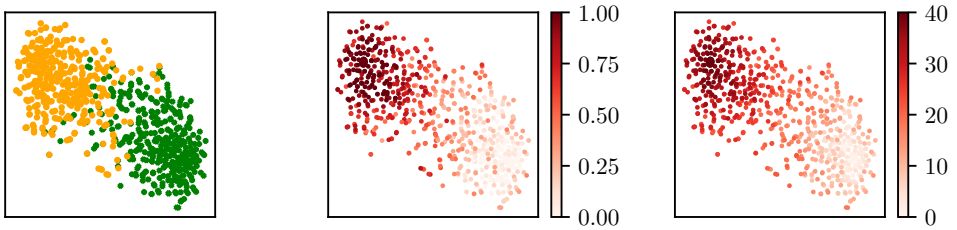


Figure 6: Hard and soft clustering output using the post-processings described in Section 3.4. We used tSNE to embed the questionnaire participants into two dimensions. The left figure shows the output of the hard clustering post-processing. The middle image shows the soft clustering for tangle A, and the right figure visualizes the hand-crafted scores: s_{npi}.

clustering output (Section 3.4) and assign each participant to one cluster, labeling them as either narcissistic or not, cf. Figure 6 left. However, this approach is very restrictive, and given our knowledge about the data, namely that there are scores on a large range and not binary classes, it seems inappropriate. Instead, we are interested in a soft output assigning a probability to each participant belonging to each cluster. We calculate these probabilities by our post-processing described in Section 3.4. The result can be seen in Figure 6. We plotted the sampled subset using tSNE (van der Maaten and Hinton, 2008) to embed the points into two dimensions. The two clusters in Figure 6 (left) correspond to one tangle each, and we assign points by their probability of belonging to one or the other tangle. Figure 6 (middle) visualizes our soft clustering output that indicates the probabilities of belonging to one tangle. In this case, we plot the probabilities of belonging to τ_A, which points toward the upper left structure. In the right image of Figure 6 we visualize the score s_{npi} as a reference. Figure 7 shows the correlation between the hand-crafted score s_{npi} and the probability of being narcissistic based on the answers returned by the algorithm. The correlation coefficient is again very high, with a value of 0.944.
Looking at the tangle search tree, we can now calculate the characteristic cuts that help distinguish between the two clusters. The splitting tangle has eight characteristic cuts,

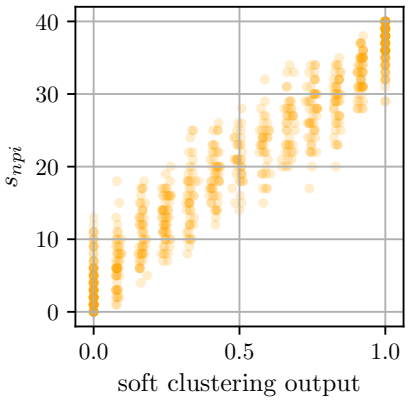


Figure 7: Correlation of the true score function and the deviation from the found tangle.

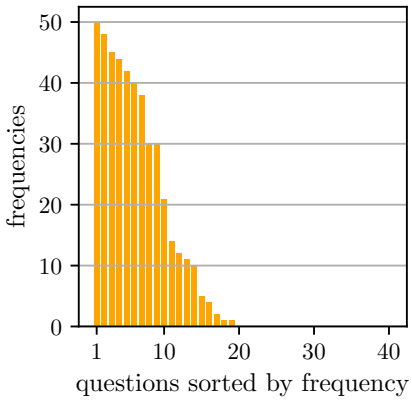


Figure 8: Frequencies of the important questions within 50 runs on random balanced sub-samples of the data.

meaning eight questions are essential to separate the two dense structures. Note that we can get variation in these results due to balanced sub-sampling, and the above shows one possible example. To support our claims, we ran the same procedure 50 times. Each time we sampled a random balanced subset of the data. The algorithm identifies between minimal five and maximal 11 important questions. We take their union, which results in an overall of 18 questions that seem to be important for splitting the data. This shows that the important questions overlap and underpins the claim that there are questions of little interest for the task. Figure 8 shows the frequencies of questions and Table 1 lists the 5 most important statements which where among the characterizing ones in at least 42 of the 50 runs. We list all statements of the dataset in Appendix III.3.

#	question number	statement A	statement B
50	12	I like to have authority over other people.	I don't mind following orders.
48	11	I am assertive.	I wish I were more assertive.
45	5	The thought of ruling the world frightens the hell out of me.	If I ruled the world it would be a better place.
44	31	I can live my life in any way I want to.	People can't always live their lives in terms of what they want.
42	32	Being an authority doesn't mean that much to me.	People always seem to recognize my authority.

Table 1: Characterizing question. Most left column gives the number of occurrences of the question within 50 runs. The question number is the one given by the dataset.

4.2 Theoretical guarantees: Binary Questionnaire

This section proposes a generative model to simulate mindsets in binary questionnaires. Based on this model, we prove that for suitable parameter choices, tangles *recover* the mindsets; that is, the set of all tangles coincides with the set of all mindsets with high probability. We refer to Appendix III.1 for the proofs.

4.2.1 Generative model

We simulate n persons that answer a questionnaire with m questions. We start by generating k ground truth mindsets $\mu_1, \dots, \mu_k \in \{0,1\}^m$. Each vector μ_i describes one specific way of answering all m questions, so it represents the stereotype person with the corresponding mindset i. We generate the entries of each ground truth mindset vector by independent, fair coin throws. For every μ_i, we generate a corresponding group V_i of n/k persons. We now choose a noise probability $p \in (0, 0.5)$ and let every person $v \in V_i$ answer question s as indicated by $\mu_i(s)$ with probability $1-p$ and give the opposite answer with probability p (independently across questions and persons). The union of the groups then forms the total population $V = \bigcup_{i=1}^{k} V_i$. Based on the answers, each question s induces a cut of V into the set $A_s^0 = \{v \in V \mid v(s) = 0\}$ and its complement $(A_s^0)^{\complement} = A_s^1 = \{v \in V \mid v(s) = 1\}$, where $v(s) \in \{0,1\}$ denotes the answer of person v to question s; the collection of these cuts is denoted by $\mathcal{P}$. Since the questions induce the cuts, there is a natural one-to-one relationship between orientations of $\mathcal{P}$ and vectors in $\{0,1\}^m$. Using this relationship, we say that the tangles recover the mindsets if the set of all $\mathcal{P}$-tangles coincides with the set of all mindsets $\{\mu_1, \dots, \mu_k\}$.

When sampling the ground truth mindsets, we need to ensure that the vectors μ_i are not degenerate because the vectors by accident support more than k tangles. We discuss the corresponding non-degeneracy-condition in Appendix III.1 (Assumption 1), where we also prove that this condition is satisfied with high probability.

4.2.2 Main result in the questionnaire setting

The following theorem states that the orientations induced by the ground truth mindsets give rise to tangles and that all tangles on $\mathcal{P}$ correspond to mindsets with high probability.

Theorem 2 (Tangles recover the ground truth mindsets) *Assume that the model parameters n, m, k and p and the tangle parameter a satisfy $p < 1/(k+3)$ and $a \in (pn, (1-3p)n/k)$. Let $\mathcal{P}$ be the set of cuts induced by questions in the questionnaire. Then with high probability, the mindsets correspond to tangles:*

1. *The probability that at least one of the mindsets does not induce a tangle is upper bounded by $km \exp\left(-2n(ka/n - 1 + 3p)^2/9k\right)$.*
2. *If the non-degeneracy Assumption 1 holds for the ground truth tangles, then the probability that there exists a spurious tangle that does not correspond to one of the mindsets is upper bounded by $km \exp\left(-2n(a/n - p)^2/k\right)$.*

In both statements, we take the probability over the random draw of the person's answers (and not over the randomness in generating the ground truth mindsets, which only play

a role regarding Assumption 1). In particular, the probability that the set of mindsets corresponds precisely to the set of tangles tends to 1 as n tends to ∞ with fixed a/n.

The theorem is based on some conditions. The bound on p ensures that the noise is not too large, considering the number of clusters. If the noise is too high, it becomes difficult to distinguish small clusters from spurious noise. The agreement parameter a must not be too small (so that we do not cluster on noise) and not too large considering cluster size (otherwise, we cannot find the clusters anymore).

This theorem is what we would like to achieve: unless the parameters are so that they obfuscate the cluster structure, tangles provably find the ground truth clusters.

4.3 Experiments on synthetic data: Binary Questionnaire

We now run experiments on the generative model described in Section 4.2. We evaluate the influence of noise in the answers and the influence of irrelevant questions, that is, questions answered at random, and compare the performance of our post-processing to the output of the k-means algorithm.

If not stated otherwise, in our algorithmic setup, we use the bipartitions induced by all questions and choose a to be a 1/3 of the size of the smallest cluster. We choose the average Hamming similarity, stated in Equation (4), to assign a cost to the bipartitions. We use the normalized mutual information (nmi) between the ground truth mindsets and our discovered mindsets to measure the performance of our hard clustering output. The nmi assigns a score between 0 and 1; high scores indicate promising results. We average the results over ten random instances of the proposed model.

4.3.1 Tangles discover the true mindsets and perform well on noisy data

One of the properties of tangles is their soft definition using sets of three orientations. As a result, they orient all bipartitions towards dense structures while the intersection of all cuts might be empty. As discussed above, we can interpret a tangle as one specific way of answering (all) questions. This scenario represents a stereotypical way of answering the questions, while no person in the dataset has to answer in this specific way. Thus tangles are inherently able to deal with noisy data. In Figure 9, we visualize the robustness of tangles on noisy data. In our model, we simulate noise by randomly flipping a percentage of each participant's answers individually. As a result, the respective person deviates more from the stereotypical answers, thus from its ground truth mindset. As a clustering baseline, we apply the k-means algorithm to the answer vectors of the participants, interpreting them as points in a Euclidean space. We give the actual number of mindsets k to the k-means algorithm. In the left image of Figure 9, we observe that for balanced datasets, tangles perform comparably to k-means. Without fine-tuning any parameters, this is significant since tangles do not directly get the number of clusters as input; only a very rough lower bound on the size of the smallest cluster we want to discover. Tangles discover the correct number of clusters and the underlying structure even with high noise in the data.

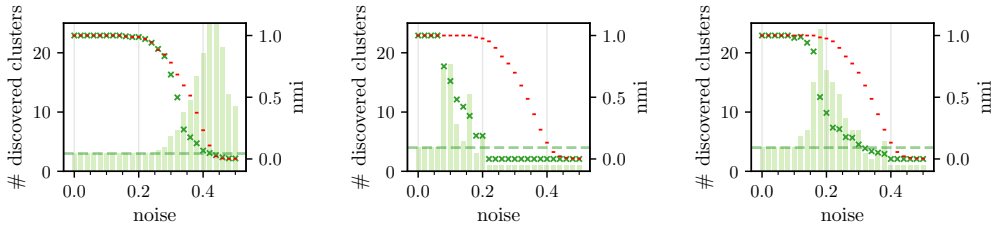


Figure 9: Influence of noise. We consider a questionnaire with 40 useful and no useless questions. We plot the performance depending on the noise for three clusters with 333 points each in the left image and show results for four unbalanced clusters with 100, 200, 300, and 400 data points in the middle and the right plot. In the two first images, we chose an agreement parameter a of 1/3 of the smallest cluster size. On the right plot, we chose a to be 2/3 of the smallest cluster size. In all three settings, we prune paths of length 1. Performance is measured by the nmi and plotted as markers. Red markers show the performance of k-means, and green markers the performance of tangles. Bars indicate the number of tangles in the data, and we evaluate both values for different noise levels in the x-axis.

4.3.2 The tangle search tree holds all the information

For unbalanced datasets, we observe one of the open problems when translating tangles into practice; the gap problem discussed in Appendix II.2.1. In a nutshell, the gap problem arises from the fact that we never consider all possible bipartitions in practice. We get a sorted subset of all possible cuts that might not cover the set of data points uniformly. Therefore, we might have gaps or large jumps between the cost of cuts – for example, many unbalanced bipartitions followed by a random cut. This phenomenon becomes especially visible in datasets that consider highly unbalanced but non-hierarchical settings, where the clusters differ significantly in density. The middle image of Figure 9 shows the performance of tangles compared to k-means. We observe that, with increasing noise, the algorithm discovers significantly more tangles than there are clusters before the number of found clusters quickly drops to one.

We can reduce the influence of these gaps by adjusting the agreement parameter or the threshold Ψ (see also Appendix II.2.1, Section 3.5). However, fine-tuning the parameters is not the goal in the end, and we believe there are other methods of post-processing the tangle search tree to avoid this, such as pruning. To highlight that tangles can also yield better results, and the tangle search tree holds all the information, we ran the same experiment with a tighter bound on the size of the smallest cluster. The larger agreement parameter results in the tangle search tree becoming inconsistent earlier and reassembles to early stopping the algorithm or choosing a smaller maximal order Ψ. In this case, we set the agreement parameter a to $\frac{2}{3}$ the size of the smallest cluster. The left plot of Figure 9 shows the improvement when better estimating a, proving that the hierarchy of the tangles search tree contains the correct cluster structure.

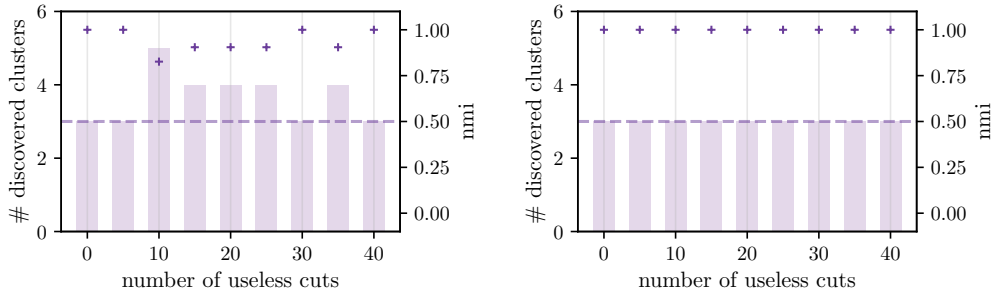


Figure 10: Influence of useless questions (answered at random) with and without pruning. We consider three clusters with 333 datapoints each. The noise p is set to 0.15 and the questionnaire has 40 useful questions. The plot shows results on two axes. The left y-axis indicated by pluses visualizes the performance measured by the the nmi. Additional on the right y-axis we show the number of tangles as bars. Both are plotted over the number of useless questions. Results in the left image show output without any pruning. Random questions can result in more tangles than there are clusters. In the right image we visualize results when pruning path of length 1. Pruning can neutralize the effect of random questions.

4.3.3 Useless questions do little harm

A bipartition is useless when it does not contain information relevant to separating the cluster structure; for example, a bipartition which roughly separates all the clusters in half. In this setting of a binary questionnaire, these would be questions that are irrelevant to the considered topic. We assume that for a given topic, such as narcissism in the npi data of Section 4.1, the question, "Do you wear glasses?" will not give any insight into a person's narcissism. However, the question is whether such useless bipartitions influence the quality of the tangle algorithm. In theory, such random bipartitions do not hurt tangles since the cut-cost will be high, so the cuts will be oriented late or not at all. They either get forced to one orientation by previous cheap bipartitions, or the tangle search tree gets inconsistent before orienting them.

As mentioned above, in practice, due to a lack of richness in the subset of cuts, we might find a large gap (cf. Appendix II.2.1) in the cost function between two cuts. Thus, even if the subsequent cut is useless, the cut will still be consistently orientable in both directions and result in two tangles splitting the large cluster into two smaller ones. Since these useless bipartitions often have a high cost, the following bipartitions will be even higher in cost and hold little to no information. We say such a split is random, and we can leverage this information to discover those splits and prune the tree along these branches. The exact procedure is described in Section 3.4. We show the result in Figure 10. On the left, we show the results when running the algorithm without pruning. The right image visualizes the results on the same data, but in this case, we prune paths to leaves of the depth of 1. We can significantly improve the output and thus find the correct number of tangles, each

corresponding to a cluster. Using the normalized mutual information score, we evaluate the performance based on our hard clustering (see Section 3.4).

4.4 Summary: Binary Questionnaire

We showed that tangles could automatically do what psychiatrists did by hand in a real-world example. They simultaneously discover the structure of the data and give insight into the questions we ask in the questionnaire. Theoretical guarantees reveal that tangles discover the ground truth with high probability in our questionnaire model. In experiments, we investigate different properties of tangles; we consider the effect of pruning the tangle search tree and the tangles' behavior on noisy data. Even though tangles do not need the correct number of clusters as an input, tangles often perform comparably to k-means, which we initialized with the correct k. The algorithm performs well for unbalanced datasets but seems more prone to noise. By sensitive sampling or adapted post-processing, we can extract more information from the data and enhance the performance. Stressing this, we show that the tangle search tree holds more information than we can currently leverage. This indicates rewarding research directions for developing and improving the hard and hierarchical clustering algorithm. Improving the post-processing or advancing the evaluation of the tangle search tree is promising.

5. Use Case: Graphs

In graph clustering, we are given a graph and want to divide the nodes of the graph into clusters such that sets of highly connected nodes are within the same clusters and there are only a few connections between different clusters. Tangles serve as an aggregation method for a set of cuts. We can generate these cuts by fast heuristic algorithms producing weakly informative cuts of the cluster structure.

5.1 Theoretical guarantees

In the following, we analyze the theoretical properties of tangles in graph clustering in the expected graph of a stochastic block model. We refer to Appendix III.2 for the proofs.

5.1.1 Model

We consider a stochastic block model on a set V of n vertices that consists of two equal-sized blocks V_1 and V_2, which represent the ground truth clusters. Edges between vertices of the same block have weight p, and edges between blocks have weight q, where $0 \leqslant q < p \leqslant 1$. In a standard stochastic block model, we would now sample an unweighted, random graph from this model, where we would choose each edge with the probability given by p and q according to the ground truth model. In our case, we will perform the analysis just in expectation, as a proof of concept. This means we do not sample a random graph but consider the weighted graph described above.

We consider tangles induced by the set of all possible cuts $\mathcal{P}$ of the set V. We use the cost function

$$c(\{A, A^{\complement}\}) := \sum_{u \in A, v \in A^{\complement}} w(u, v) \tag{5}$$

where $w(u, v)$ denotes the weight of the edge between vertices u and v. If we denote $\alpha_i = |A \cap V_i|/|V_i|$, this gives us, as $|V_1| = |V_2| = n/2$ the following explicit formula for the cost:

$$c(\{A, A^{\complement}\}) = \frac{n^2}{4} \left(p(\alpha_1 - \alpha_1^2 + \alpha_2 - \alpha_2^2) + q(\alpha_1 + \alpha_2 - 2\alpha_1\alpha_2) \right). \tag{6}$$

Each of the two ground truth blocks V_1, and V_2 induces a natural orientation of the set of all cuts by picking from each cut $\{A, A^{\complement}\}$ the side containing the majority of that block's vertices. We find that for reasonable choices of p, q, and a, there is a range of costs in which these two orientations are indeed distinct tangles.

5.1.2 Main results in the graph clustering setting

The following Theorem states that in the graph clustering setting, tangles perfectly recover the ground truth: there exists a one-to-one correspondence between the tangles and the ground truth blocks.

Theorem 3 (Tangles recover the ground truth blocks) *Assume that the block model parameters p, q, n and the tangle parameter a satisfy $p > 3qn/(n-2a)$ and $a \geqslant 2$. Consider the set $\mathcal{P}$ of all possible graph cuts, and the set $\mathcal{P}_\Psi$ of those graph cuts with costs (cf. Equation 6) bounded by Ψ. Let $\xi = 1 + q/p$. If Ψ satisfies*

$$q\left(\frac{n}{2}\right)^2 \leqslant \Psi < \frac{n^2}{4} p \left(\frac{1}{3}\xi \left(\xi - \frac{2a}{n} \right) - \frac{1}{9} \left(\xi - \frac{2a}{n} \right)^2 \right),$$

then the two orientations of $\mathcal{P}_\Psi$ induced by the two ground truth blocks are distinct and exactly coincide with the $\mathcal{P}_\Psi$-tangles.

If, on the other hand, $p < 2q$, there is no chance that tangles identify the two blocks as distinct clusters. The intuitive reason is that the within-cluster connectivity p is smaller than two times the between-cluster connectivity q, the expected cost of a cut separating the two clusters is higher than one cutting through the clusters, which makes it impossible to recover the block structure. In this case, there will be precisely one tangle.

Theorem 4 (Non-identifiability) *If $a \geqslant 2$ and $p < 2q$, then for any value Ψ and $\mathcal{P}_\Psi$ the set of all cuts of cost at most Ψ, there exists at most one $\mathcal{P}_\Psi$-tangle.*

Note that all our results are proved in the expected model, and they assume that tangles are constructed on the set of all possible graph cuts. In the experiments in the following section we complement these results with the cases where clusters are sampled from the model and where the tangles are constructed on a realistic, small subset of graph cuts.

5.2 Experiments on synthetic datasets

To validate tangles in the graph clustering setup, we perform experiments on synthetic data where we randomly sample graphs from a standard stochastic block model.

5.2.1 Setup of the simulation and baselines

As opposed to the questionnaire setting, in the graph clustering setting, there is no obvious choice for the initial partitions in the pre-processing step. Instead, we use the Kernighan-Lin-Algorithm (Kernighan and Lin, 1970) to generate a small set of initial cuts. This algorithm performs a local search for a cheap cut under fixed partition sizes. Starting with a randomly initialized cut, each iteration goes over all pairs of vertices and greedily swaps their assignment if this improves the current cut. In the original version, the algorithm stops when none of the possible pairs can improve the cut value. However, we found that it is enough to run the algorithm for just two iterations of the local search to speed up the pre-processing: a highly diverse set of initial cuts is essential for our purpose. We denote this version of the algorithm as the KL algorithm with early stopping. Given a graph with n vertices, each pass of the algorithm runs in time $\mathcal{O}(n^2 \log n)$, and we run the algorithm for two iterations. We use the average cut value to assign a cost to each bipartition: $c(\{A, A^\complement\}) \coloneqq \frac{1}{|A|\cdot(n-|A|)} \cdot \sum_{u\in A, v\in A^\complement} w(u,v)$ where $w(u,v)$ is one if there is an edge between the nodes u and w, else 0. We then apply the tangle algorithm to the subset of bipartitions. We choose the agreement parameter for the algorithm to be 1/3 of the size of the smallest cluster, which is a rough lower bound. We do not choose a threshold value for Ψ for the tangle algorithm but use all bipartitions generated by the pre-processing. To derive a hard clustering from the tangle search tree, we apply the post-processing described in Section 3.4. To evaluate the output, we use the normalized mutual information score (nmi) and average the values over ten random instances of the stochastic block model.
As a baseline, we compare tangles to normalized spectral clustering in the *sklearn* implementation. It gets the correct number of clusters as input.

5.2.2 Tangles meet the theoretical bound already with few, weak initial cuts

The theoretical results above show that tangles recover the correct blocks in expectation based on all possible graph cuts. This section explores how far the bounds hold when we only generate a few initial cuts in a pre-processing step.
Figure 11 shows the results for (top row) two and (bottom row) five different clusters and varying values for the within cluster connectivity p and the between cluster connectivity q. In the left figures, we see the results for 20 cuts generated with the KL-Algorithm stopping after only two iterations. As indicated by the red line, tangles meet the theoretical bound in this setting. Improving the set of initial cuts by running the KL-Algorithm for 100 iterations (which usually is until convergence) and using a more significant number of cuts (100) improves the results but is barely visible. We visualize the results for this setting in the middle pictures of both rows. Tangles can only aggregate the information in the set of cuts. We perform better when the quality of the initial bipartitions increases. However, minimal improvement indicates that fast and simple algorithms usually suffice to achieve satisfying results.

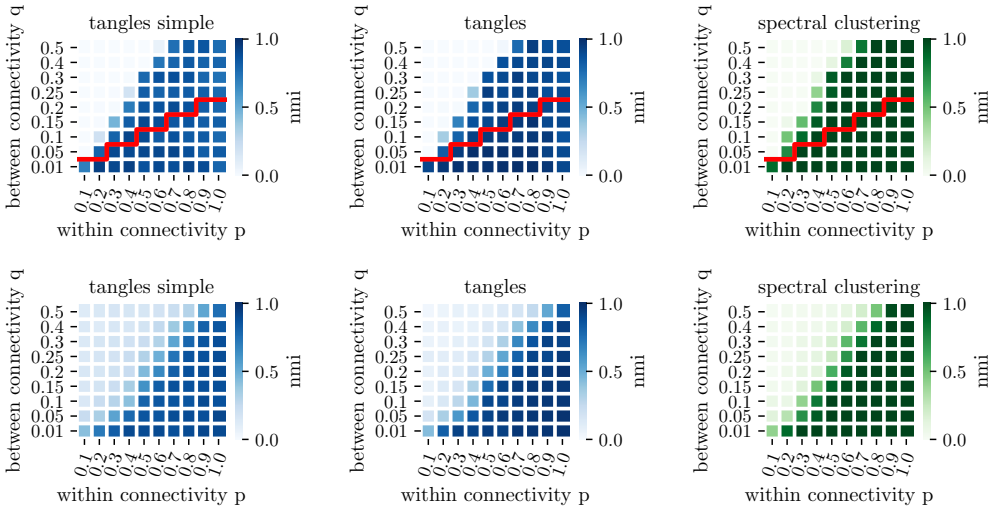


Figure 11: In the top row, we consider two clusters with 50 data points, each with an agreement parameter of $a = 16$. In the bottom row, we show results for five clusters and 20 datapoints each and an agreement parameter of $a = 6$. In the left and the middle figure, we use the KL-Algorithm to generate the initial set of cuts and apply the tangle algorithm. We achieve high performance with only 20 low-quality cuts (left). Increasing the number and the quality of the cuts to 100 and stopping after 100 iterations improves the overall performance of tangles only marginally (middle). In the right figures, we plot the results of normalized spectral clustering, which gets the correct number of clusters as a parameter. Tangles perform comparably. The red line indicates the theoretical bound derived in Section 5.1.2

Comparing the tangle results to spectral clustering, we can see that they perform comparably: they both recover the block structure under similar parameter settings and with comparable accuracy. We find this quite impressive, considering the "quick and dirty" pre-processing of generating only 20 cuts using a local search heuristic.

5.2.3 Performance of tangles saturates fast with increasing number of cuts

In the section above, we already saw that a small set of cuts slightly better than random is sufficient to yield satisfying results. In the next experiment, we investigate the number of cuts more closely. We show that the performance saturates fast with an increasing number of cuts. This observation is comforting: the number of cuts is no complex parameter to fine-tune. Figure 12 shows two simple examples for two and five clusters. With an increasing number of cuts, the performance increases fast before saturating. While for a small set of cuts, sometimes more tangles than clusters exist, with a more significant number of cuts, this number also stabilizes quickly.

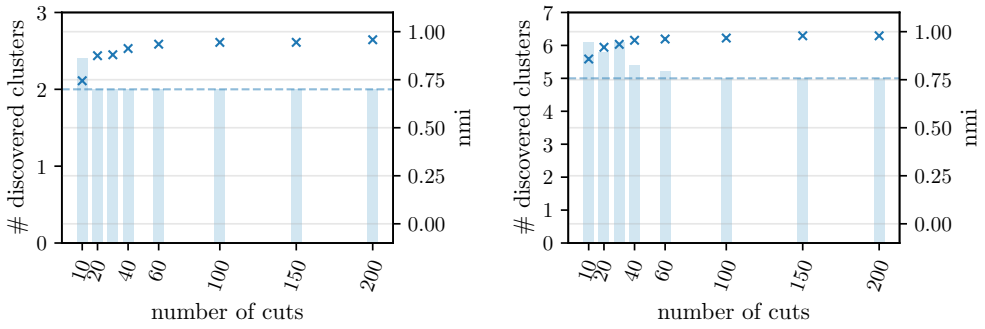


Figure 12: Performance saturates fast. (Left) Two clusters with 500 datapoints each and an agreement parameter of $a = 166$. (Right) Five clusters with 200 data points each and an agreement parameter of $a = 66$. On the y-axis, we plot the performance measured by nmi as markers, and we plot bars to visualize the number of tangles. The dashed line indicates the correct number of clusters. The number of cuts is on the x-axis. All results are averaged over ten random samples. With an increasing number of cuts, the performance increases fast before saturating.

5.3 Summary

This section demonstrates that tangles are well suited for a graph clustering setting. We provide theoretical performance guarantees and show that the algorithms work straightforwardly in practice. It is particularly encouraging to see that only a few initial cuts are necessary to achieve good performance. The number of considered cuts, which we initially believed to be the bottleneck of the computation (due to its cubic contribution to the running time), is not a limiting factor in practice. In Section 6.2, we will investigate the overall runtime of the algorithm and see that it behaves almost linearly in the number of cuts.

6. Use Case: Feature based data and interpretability

As our final use case, we consider a feature-based setup. Consider a set of data points V described in terms of a vector of features. Each dimension represents one feature; these can be categorical or binary, or continuous features. The goal is to group points into clusters so that points that are featurewise similar to each other get assigned to the same cluster, while very dissimilar points are supposed to be in different clusters. Like in the graph setting, we can use fast and randomized algorithms to compute the initial set of cuts. One example of a cut-finding algorithm in a Euclidean setting is the following heuristic: randomly project the dataset on a one-dimensional subspace and generate a bipartition by applying the 2-means algorithm.

In order to explore yet another strength of tangles, we would like to focus on interpretable clustering algorithms in this section. To this end, we generate axis parallel data cuts in our pre-processing step. We then use the tangle mechanism to reveal clusters in the data.

Algorithm 2: Generate the initial set of cuts

Data: Set of points V, agreement parameter a
Result: Set of cuts $\mathcal{P}_a$

Choose x_1 such that $|A_{x,1}| = 1$ and $|A^{\complement}_{x,1}| = n - 1$.
$\mathcal{P}_a = \{A_{x,1}, A^{\complement}_{x,1}\}$
i = 1
while $A^{\complement}_{x,i} > a - 1$ **do**
    Choose x_{i+1} such that $|A^{\complement}_{x,i} \cap A^{\complement}_{x,i+1}| = a - 1$
    $\mathcal{P}_a.append(\{A_{x,i+1}, A^{\complement}_{x,i+1}\})$
    i = i+1
end
return $\mathcal{P}_a$

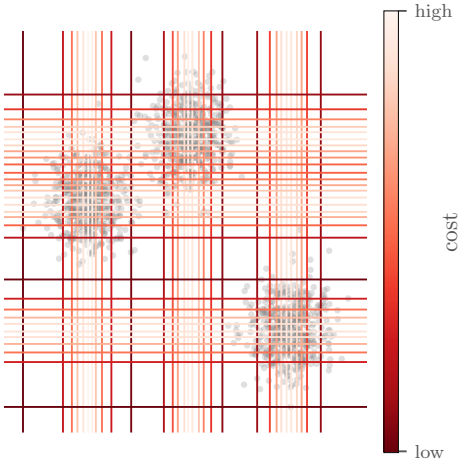


Figure 13: Visualization of resulting cuts.

These clusters then have a simple description in terms of features. Similar procedures have been used in interpretable clustering; see related work, Section 7.

6.0.1 Setup of our interpretable tangle framework

Consider the set $V \subset \mathbb{R}^d$ (we assume all points are pairwise different). We generate axis-parallel cuts by a simple slicing algorithm. Moving along each axis, we select cuts exactly $a-1$ points away from each other. We outline the details in the pseudocode in Algorithm 2. Here $A_{x,i}$ represents the set of points smaller than some real value x_i along the x-axis. $\{A_{x,i}, A^{\complement}_{x,i}\}$ is the cut along the x-axis at the real value x_i. The algorithm computes $\mathcal{O}(n/a)$ cuts for each dimension. The complexity is linear in the number of points and the number of dimensions: $\mathcal{O}(n \cdot d)$. As in the settings above, one possible post-processing is the one we describe in Section 3.4, which gives us a hard clustering output. If we have a low-dimensional embedding of our data, we can nicely visualize the soft output like in Figure 2.

6.1 Theoretical guarantees in the feature-based setting

We now prove theoretical guarantees for the tangle algorithm in the feature-based setting. As a ground truth model, we use a mixture of Gaussians. All theoretical results build on the pre-processing with axis-parallel cuts.

6.1.1 Ground truth model: a simple Gaussian mixture

Suppose we are given two cluster centers $\mu = (\mu_1, \dots, \mu_d)$ and $\nu = (\nu_1, \dots, \nu_d)$ as points in the d-dimensional space. For ease of notation, let us assume that $\mu_i \leqslant \nu_i$ for all i. We

suppose that our data points V are obtained by sampling n points in total from a mixture of two Gaussian distributions $\mathcal{N}(\mu, \sigma^2 I)$ and $\mathcal{N}(\nu, \sigma^2 I)$ with equal weight, one with center μ and one with center ν, and each with variance $\sigma^2 I$. Let us denote the bipartitions of V obtained from Algorithm 2 along dimension j as $(A_{j,1}, A_{j,1}^{\complement}), \dots, (A_{j,m}, A_{j,m}^{\complement})$, where $A_{j,i} \subseteq A_{j,i+1}$. Let us, for the moment, assume that we sampled all axis-parallel bipartitions, that is, we used $a = 2$ for our sampling algorithm. Moreover, let us denote $x_{j,i}$ as the point in $\mathbb{R}$ for which we obtained $(A_{j,i}, A_{j,i}^{\complement})$ as $A_{j,i} = \{v \in V \mid v_j < x_{j,i}\}$. The set of all the bipartitions $(A_{j,i}, A_{j,i}^{\complement})$ for a fixed j is denoted as $\mathcal{P}_j$.
For our proofs, we work in a scenario "in expectation": whenever we need to compute the volume of a set, we use the expected volume rather than the volume induced by the actual sample points.

6.1.2 Main results

We will show that, under favorable conditions, there are two tangles, each pointing to one of the cluster centers μ and ν, respectively. Here, the side of a cut $\{A_{j,i}, A_{j,i}^{\complement}\}$ that *points to* $y \in \mathbb{R}^n$ is $A_{j,i}$ if $x_{j,i} > y_j$ and $A_{j,i}^{\complement}$ if $x_{j,i} < y_j$; an orientation of cuts *points to* y if all the sides of all cuts points to y.
The following theorem says that if the distance between the cluster centers is large enough and the agreement parameter a is small enough, then we find at least two different tangles: one pointing to μ and one pointing to ν.

Theorem 5 (All cluster centers induce distinct tangles) *Let $a < n/12$. If along some axis j there is a local minimum $(A_{j,i}, A_{j,i}^{\complement})$, which is a global minimum and whose location $x_{j,i}$ has distance more than σ to both μ_j and ν_j, then there exist (at least) two tangles τ_μ and τ_ν, where τ_μ points to μ but not ν and τ_ν points to ν but not μ.*

The following theorem states that there are no spurious extra tangles if a is chosen large enough, whereas this bound becomes lower the further apart μ and ν are.

Theorem 6 (All tangles point to distinct cluster centers) *Let q be at most the fraction of points from ν at distance* dist*; $q \leqslant (1 + \mathrm{erf}(-\mathrm{dist}/(2\sqrt{2\sigma^2})))/2$. If there exists a dimension j where* $\mathrm{dist} = |\mu_j - \nu_j|$ *is large enough, that is* $\mathrm{dist} > 2\sigma$*, then for $a > n \cdot (0.42q + 0.056)$, every tangle points to either μ or ν.*

The bound in Theorem 6 meets the one from Theorem 5 for $\mathrm{dist} \gtrsim 3,03\sigma$. In practice, we observe that the bounds are not tight, and the range in which we can choose the agreement parameter a is much larger. We do not investigate the range for which the algorithm returns the perfect results; in practice we found the agreement parameter to be easy to choose. Usually, a rough estimate of the smallest cluster we want to discover suffices.

6.2 Experiments on synthetic datasets

In this section, we run experiments on a simple instance of a mixture of Gaussians, as the one shown in Figure 2. To generate an initial set of bipartitions, we use the slicing Algorithm 2 described in Section 6.0.1. As a cost function, we use

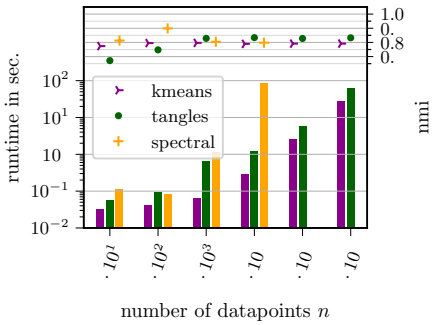


Figure 14: Performance and runtime of tangles with $m = 40$ cuts on a mixture of 4 Gaussians in two dimensions are competitive to two baselines. Runtime is shown as bar plots, and nmi performance with markers. We ran each algorithm for at most 1 hour; spectral clustering was too slow for more than 60.000 data points.

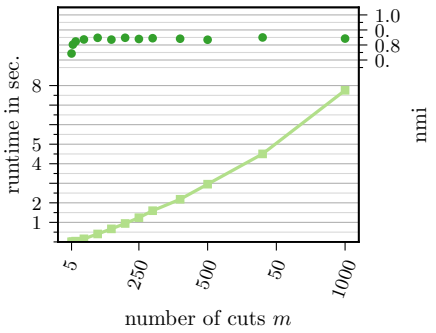


Figure 15: Computation time for the tangle search tree on a mixture of 4 Gaussians in two dimensions with $n = 6,000$ data points. Time grows less than cubic in the number of cuts, and performance quickly saturates.

$$c(\{A, A^{\complement}\}) = \sum_{v \in A, u \in A^{\complement}} -||v - u||. \tag{7}$$

To make the different methods comparable, we consider a challenging clustering task in which no method predicts the ground truth clusters. We assess their performance via the normalized mutual information between predicted and actual clusters. We experiment on simple instances of a mixture of Gaussians in $\mathbb{R}^2$ with $n = 6,000$ points and $k = 4$ clusters as the one visualized in Figure 2. All results are averaged over ten random instances of the model. We compare tangles to the k-means clustering algorithm as implemented in *sklearn*. We consider the agglomerative method of average linkage and a divisive method as hierarchical methods. For the latter, we iteratively use spectral clustering to split the cluster with the largest number of points into two clusters. All baseline algorithms get the correct number of ground truth clusters as an input parameter; tangles do not need this — they only get a rough lower bound on the size of the smallest cluster, specified by the agreement parameter a.

6.2.1 Computing tangles is fast

On the mixture of Gaussians, tangles perform comparable to k-means — although we used a straightforward cut-finding algorithm. Spectral clustering performs consistently well but is significantly slower, as shown in Figure 14.

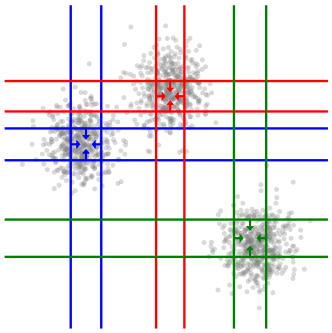


Figure 16: Visualisation of the core of all three tangles. Each color indicates one tangle. All other orientations are induced by the orientation of the core by the consistency condition.

Figure 17: Visualization using the soft clustering output for two dimensional data.

While the overall runtime of the algorithmic framework also depends on the pre-processing, computing the tangles itself is linear in the number of data points. In Figure 15, we investigate the complexity concerning the number of cuts. As discussed in Section III.2.1, the worst-case complexity is cubic in the number of cuts m. However, our experiments show that this bound is quite pessimistic because many branches quickly become inconsistent, and additionally, the performance already saturates after a few cuts. Hence, this experiment demonstrates that the number of cuts is not the limiting factor of tangles, similar as we have seen it in the graph cut setting as well.

Dataset	Mixture of Gaussians
Linkage	0.664
Divisive	0.820
k-Means	0.797
Spectral	0.805
Tangles	**0.829**

Table 2: Performance across all methods measured by normalized mutual information and averaged over ten runs.

6.2.2 Interpretability of cuts translates to interpretable clustering

If the initial bipartitions are interpretable, so are the resulting clusters. The small number of necessary cuts enhances this effect. We have already seen this in the questionnaire setting, where tangles find a small number of essential questions that can characterize the clustering sufficiently. Now we look at interpretability in the feature-based setting. Axis parallel cuts are interpretable: "all patients with temperature larger than 39 Celsius". We can carry over such explanations to tangles in the following way.

A tangle gives a consistent orientation to a set of bipartitions. If every cut is interpretable, we can combine these interpretations with an "and". In a tangle, there will likely be redundant bipartitions in the sense that two bipartitions point towards the same direction along the same axis, but one is more restrictive. For the explanation, we only consider the most restrictive bipartitions along the axes. We then end up with an interpretation that gives us an interval on each of the dimensions: dimension d_1 between x_1 and y_1 **and** dimension d_2 between x_2 and y_2 and so on. This interpretation represents the core of our tangle and points to the center of the respective cluster. Based on this, we can explain the cluster; there is a cohesive structure with these properties. Note that it does not follow that points outside of this 'core' do not belong to the same structure. This fact arises from the soft definition of tangles. Based on the tangle search tree, we can develop different approaches to interpret the resulting tangles depending on what we aim to characterize. Using the hard clustering algorithm, we can define the boundary cuts of each of the clusters or find the intervals of indistinct points, that is, points that belong to two (or more) clusters with comparable probabilities, as well as the characterizing cuts that distinguish between two clusters. Figure 16 visualizes an interpretation of the core in the 2-dimensional setting. For data embedded in two dimensions, a heat-map of our soft clustering already gives a visual interpretation of the tangles. Figure 17 shows an example for the used mixture of Gaussians.

6.3 Summary of the feature-based scenario

We used a naive pre-processing in the feature setting: axis parallel cuts. Even with these simple cuts, tangles perform comparably to the baseline clustering algorithms in terms of clustering accuracy while at the same time predicting the number of clusters in the data. As we can see, tangles can be computed fast, similarly to the k-means algorithm, and far faster than spectral clustering. We also showed that tangles could allow for natural interpretations in some cases.

7. Related work

Clustering is a vast field that comes with a multitude of algorithms. Conceptually, there are some related lines of work that we briefly want to touch on in order to position our framework and the tangles background into the landscape of clustering algorithms.

Clustering ensembles. Clustering ensemble methods first generate a set of initial clusterings using multiple clustering algorithms and then combine them with a consensus function to form a final clustering (Vega-Pons and Ruiz-Shulcloper, 2011). Even though this roughly resembles the tangle framework, there are key differences. In particular, ensemble methods typically use "strong" clustering algorithms that provide close-to-perfect clusters already. The ensemble mechanism is then only invoked to make the result more robust against the biases of the individual methods. The tangle philosophy is quite different: we use "quick and dirty" heuristics to produce the initial cuts, which then get boosted to high-quality clusterings.

Soft clustering. Soft clustering relaxes the degree to which objects belong to clusters from unique assignments (hard clustering) to distributions over the clusters, see, for example, Hastie et al. (2009). Similar to soft k-means, we can interpret tangles as representing clusters by abstract cluster centers, which we can convert into soft clustering. While soft clustering algorithms specify the number of clusters in advance, tangles indirectly specify a cluster's minimum size through an agreement parameter.

Hierarchical clustering. In hierarchical clustering, we cluster a dataset at different granularity levels, often represented by a dendrogram (Murtagh and Contreras, 2017). The tangle search tree also encodes a hierarchical structure, and its generation strongly resembles divisive approaches. Fluck (2019) even showed that in one specific setup considering all possible cuts, tangles and single linkage hierarchical clustering coincide. In general, however, there are fundamental differences between the two: the subdividing cuts are given and not computed recursively, the nodes represent tangles and not subsets, and the consistency condition provides a natural stopping criterion.

Interpretable and Explainable Clustering. To date, there is little work on explainable unsupervised learning as clustering. While there are papers considering decision trees for explainable clustering (Fraiman et al., 2013; Bertsimas et al., 2018), most work is empirical without theoretical analysis of the performance. Recently Dasgupta et al. (2020) and Frost et al. (2020) developed an algorithm for explaining the k-means clustering, approximating the performance in practice and theory. They do so by combining unsupervised and supervised learning first to construct a clustering and then, using the output as ground truth, explain the result using decision trees. In contrast, tangles come with an inherently interpretable output as long as the initial cuts are interpretable. For geometric data, using only axis-parallel cuts, tangles will directly return an interpretable output without further post-processing or approximating the clustering.

Theoretical results on clustering. Few clustering algorithms and generative models admit consistency guarantees: spectral clustering is among them, and its behavior on stochastic block models is well understood, see Abbe (2018) and references therein. Linkage algorithms typically are not statistically consistent (Hartigan, 1981) and thus do not necessarily discover the ground truth hierarchy, but some of them admit guarantees on approximations (A. Rinaldo, 2010; Chaudhuri et al., 2014; Moseley and Wang, 2017; Cohen-Addad et al., 2017). Guarantees for k-means are complex because of local optima, but with careful initialization, some approximation guarantees exist (Arthur and Vassilvitskii, 2007). Moreover, a large bulk of theory literature exists on Gaussian mixture clustering. To only mention a part of it, see Dasgupta (1999); Genovese and Wasserman (2000); Ghosal and van der Vaart (2001); Arora and Kannan (2005); Li and Schmidt (2015) for learning algorithms, convergence rates, and theoretical performance guarantees, see Banks et al. (2017); Ashtiani et al. (2018) for theoretical, and complexity bounds. See Vankadara and Ghoshdastidar (2020) for optimality guarantees of kernel methods in high dimensions.

Mathematical background on tangles. Tangles were initially conceived in the Graph Minors Project of Robertson and Seymour (1991) as a tool to measure how 'tree-like' a

graph is. In the original sense, Tangles are orientations of bipartitions of the *edge set*, which represents a hard-to-separate area inside a graph, making it very unlike a tree. This interpretation led to an abstraction of this notion, introducing the concept of tangles first to other contexts such as matroids (Geelen et al., 2009), and later resulted in the development of an abstract framework that unifies the notion of tangles from various contexts (Diestel and Oum, 2021; Diestel et al., 2019). Grohe (2016) performed a detailed survey on tangles for connectivity functions, a large and essential subclass of the more general tangles mentioned above. Grohe and Schweitzer (2015) created a sophisticated algorithmic framework and data structure for efficiently computing these tangles along with the corresponding tree of tangles. These works developed orthogonally to the ideas of Diestel and Whittle (2016), leading up to Diestel (2019) and Diestel (2020). Diestel focuses on making the notion of tangles applicable to as wide a range of settings as possible. To do so, he suggests softening some of the mathematical requirements of tangle theory. Their rigorous mathematical results may no longer apply to Diestel's tangles, which nevertheless aim to capture the notion of clusters. Our work in this paper follows the impulse of these latter ideas, taking them into a machine-learning context. In particular, the approach of sampling cuts, where the mathematical theory demands to consider all cuts up to a specific order, fits into this picture of approximating an underlying, more rigorous mathematical object.

8. Conclusion

In this paper, we introduce tangles to the machine learning community. This required a significant effort to simplify general concepts to convert the mathematical theory of tangles to a practical framework. We provide a first framework that works in practice and give provable guarantees in three statistical models. The general concept of "pointing towards a cluster" is a flexible formulation of a generic clustering problem. It only requires a set of cuts of the dataset and some notion of similarity between the objects. Thus tangles are directly applicable to many datasets without a workaround like building a nearest-neighbor graph or embedding the nodes. Although we convert the output to a hard clustering for the numeric evaluation, it is of a more general soft and hierarchical nature.

Note that we do not claim that tangles outperform every other algorithm out there. However, we are intrigued by their flexibility and potential. We proved performance guarantees in three very different setups: the questionnaire setting, the stochastic block model setting, and a Gaussian mixture setting. We are aware that stronger guarantees can be proved for individual algorithms in each setting. However, we are unaware of any algorithm for which guarantees can be proved in many different scenarios. A similar statement holds for our experiments. What is impressive is that the tangles framework combines many desirable properties that none of the baseline methods can provide at the same time: it is accurate without making assumptions on the shape of the clusters (as spectral clustering), it is fast (as k-means), it generates a hierarchy (as average linkage) and can be post-processed to a soft clustering, and it entails natural explanations.

We consider this work the first proof of concept that establishes tangles as a promising tool for clustering. More future work is needed to explore the full potential. There are many open questions for future research. On the algorithmic side, what is the optimal interplay between the initial cuts, the tangle algorithm, and the post-processing? On the theoretic

side, the most intriguing question is whether it is possible to formalize the intuition that tangles provide a generic tool to convert many "weak" cuts to "strong" clusters, as is the case for boosting in classification.

Acknowledgments

This work has been supported by the German Research Foundation through the Cluster of Excellence "Machine Learning – New Perspectives for Science" (EXC 2064/1 number 390727645) and the International Max Planck Research School for Intelligent Systems (IMPRS-IS)

References

L. Wasserman A. Rinaldo. Generalized density clustering. *Annals of Statistics*, pages 2678–2722, 2010.

E. Abbe. Community Detection and Stochastic Block Models: Recent Developments. *Journal of Machine Learning Research (JMLR)*, 18(177):1–86, 2018.

S. Arora and R. Kannan. Learning mixtures of separated nonspherical Gaussians. *The Annals of Applied Probability*, 15(1A):69–92, 2005.

D. Arthur and S. Vassilvitskii. K-Means++: The Advantages of Careful Seeding. ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007.

H. Ashtiani, S. Ben-David, N. Harvey, C. Liaw, A. Mehrabian, and Y. Plan. Nearly tight sample complexity bounds for learning mixtures of Gaussians via sample compression schemes. 2018.

J. Banks, C. Moore, N. Verzelen, R. Vershynin, and J. Xu. Information-theoretic bounds and phase transitions in clustering, sparse PCA, and submatrix localization. *IEEE Transactions on Information Theory*, 64(7):4872–4894, 2017.

D. Bertsimas, A. Orfanoudaki, and H. Wiberg. Interpretable Clustering via Optimal Trees. *preprint arXiv:1812.00539*, 2018.

K. Chaudhuri, S. Dasgupta, S. Kpotufe, and U. von Luxburg. Consistent Procedures for Cluster Tree Estimation and Pruning. *IEEE Transactions on Information Theory*, 60 (12):7900–7912, 2014.

V. Cohen-Addad, V. Kanade, and F. Mallmann-Trenn. Hierarchical Clustering Beyond the Worst-Case. Neural Information Processing Systems (NeurIPS), 2017.

S. Dasgupta. Learning Mixtures of Gaussians. Foundations of Computer Science (FOCS), 1999.

S. Dasgupta, N. Frost, M. Moshkovitz, and C. Rashtchian. Explainable k-Means and k-Medians Clustering. *preprint arXiv:2002.12538*, 2020.

Clustering with Tangles

R. Diestel. Abstract separation systems. *Order*, 35(1):157–170, 2018.

R. Diestel. Tangles in the social sciences. *preprint arXiv:1907.07341*, 2019.

R. Diestel. Tangles - A new paradigm for clusters and types. *preprint arXiv:2006.01830*, 2020.

R. Diestel and S. Oum. Tangle-tree duality in abstract separation systems. *Advances in Mathematics*, 377:107470, 2021.

R. Diestel and G. Whittle. Tangles and the Mona Lisa. *preprint arXiv:1603.06652*, 2016.

R. Diestel, F. Hundertmark, and S. Lemanczyk. Profiles of separations: in graphs, matroids, and beyond. *Combinatorica*, 39(1):37–75, 2019.

E. Fluck. Tangles and Single Linkage Hierarchical Clustering. Mathematical Foundations of Computer Science (MFCF), 2019.

R. Fraiman, B. Ghattas, and M. Svarc. Interpretable clustering using unsupervised binary trees. *Advances in Data Analysis and Classification*, 7:125–145, 2013.

N. Frost, M. Moshkovitz, and C. Rashtchian. ExKMC: Expanding Explainable k-Means Clustering. *preprint arXiv:2006.02399*, 2020.

J. Geelen, B. Gerards, and G. Whittle. Tangles, tree-decompositions and grids in matroids. *Journal of Combinatorial Theory, Series B*, 99(4):657–667, 2009.

C.R. Genovese and L. Wasserman. Rates of Convergence for the Gaussian Mixture Sieve. *The Annals of Statistics*, 28(4):1105–1127, 2000.

S. Ghosal and A. W. van der Vaart. Entropies and rates of convergence for maximum likelihood and Bayes estimation for mixtures of normal densities. *The Annals of Statistics*, 29(5):1233–1263, 2001.

M. Grohe. Tangled up in blue (a survey on connectivity, decompositions, and tangles). *preprint arXiv:1605.06704*, 2016.

M. Grohe and P. Schweitzer. Computing with Tangles. *Symposium on Theory of Computing (STOC)*, pages 683–692, 2015.

J.A. Hartigan. Consistency of Single Linkage for High-Density Clusters. *Journal of the American Statistical Association*, 76(374):388–394, 1981.

T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer, 2009.

F. Helguerro. Sui Massimi Delle Curve Dimorfiche. *Biometrika*, (3):85–98, 1904.

B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell Systems Technical Journal*, 49(2), 1970.

J. Li and L. Schmidt. A Nearly Optimal and Agnostic Algorithm for Properly Learning a Mixture of k Gaussians, for any Constant k. *preprint arXiv:1506.01367*, 2015.

B. Moseley and J. Wang. Approximation Bounds for Hierarchical Clustering: Average Linkage, Bisecting K-means, and Local Search. pages 3094–3103. Neural Information Processing Systems (NeurIPS), 2017.

R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview, II. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(6):e1219, 2017.

T. Raskin. A principal-components analysis of the Narcissistic Personality Inventory and further evidence of its construct validity. 1988.

N. Robertson and P.D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

M.F. Schilling, A.E. Watkins, and W. Watkins. Is Human Height Bimodal? *The American Statistician*, 56(3):223–229, 2002.

L. van der Maaten and G. Hinton. Visualizing Data Using T-SNE. *Journal of Machine Learning Research (JMLR)*, 9(11):2579–2605, 2008.

L.C. Vankadara and D. Ghoshdastidar. On the optimality of kernels for high-dimensional clustering. International Conference on Artificial Intelligence and Statistics (AIStats, 2020.

S. Vega-Pons and J. Ruiz-Shulcloper. A survey of clustering ensemble algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(03):337–372, 2011.

D. Zwillinger and S. Kokoska. *CRC standard probability and statistics tables and formulae*. Crc Press, 1999.

Appendix

I. Background on tangles

I.1 Translation between our terminology and the one used in graph theory

Tangles originate in mathematical graph theory, where they are treated in much more generality than what we need in our paper (cf. Diestel (2018) for an overview). For our work, we condensed the general theory to what we believe is the essence of tangles needed for machine learning applications. Since our setting is much simpler than the general one in mathematics, we also opted for more straightforward terminology closer to the machine-learning community's language. Table 3 provides a glossary of the correspondences between our terminology and that of Diestel (2018).

This paper		**Mathematical literature, e.g. Diestel (2018)**
cuts/partitions	instance of	separations
side of a cut/oriented cut	instance of	oriented separation
cost of a cut	instance of	order of a separation
set $\mathcal{P}$	instance of	separation system S
O is consistent	corresponds to	O avoids $\mathcal{F} = \{\{A_1, A_2, A_3\} \mid \lvert\bigcap_{i=1}^{3} A_i\rvert < a\}$
O is a $\mathcal{P}_\Psi$-tangle	corresponds to	O is an $\mathcal{F}$-tangle of $\mathcal{P}_\Psi$ (for $\mathcal{F}$ as directly above)
not used: condition $A \cap B \neq \emptyset\ \forall A, B \in O$	—	O is consistent
tangle search tree *(the output of our algorithm)*	—	*not used*
not used	—	tree of tangles *(a tree-like set of cuts that partitions the tangles; similar to a space-partition-tree)*

Table 3: Translation table between this paper and the mathematical literature

Furthermore, whereas we denote by $\mathcal{P}_\Psi$ the set of all cuts from $\mathcal{P}$ of costs *at most* Ψ, in the mathematical literature, S_k usually denotes the set of all separations from S of order *less than* k.

Note that throughout the literature, there are multiple ways to denote an oriented cut. Diestel (2018), for example, comes at the topic from the angle of graph theory, closely following Robertson and Seymour (1991), and has the convention of always listing both sides $(A^\complement, A)$, which is interpreted as *pointing from* $A^\complement$ *towards* A. On the other hand,

coming from matroid theory, Grohe and Schweitzer (2015), among others, have the custom of representing a tangle as a set of all the *small sides*, the $A^{\complement}$s in our terminology, which means the element-wise complements of the tangles in this paper would represent tangles in their sense.

I.2 The magic number three in the consistency condition.

The consistency condition in Eq. (1) is defined with exactly three cuts. Three is the lowest number which makes the *profile property* in mathematical tangle theory come true: if O is a tangle of $\mathcal{P}_\Psi$ then $A, B \in O$ implies that $(A \cap B)^c \notin O$. This property is central to tangle theory; for example, in the proof of one of the two central theorems from tangle theory, the tree-of-tangles theorem (Diestel et al., 2019). As a corollary, this theorem gives us a bound on the number of tangles. If $\mathcal{P}$ is the set of all cuts of some V, and we consider sets of less than three, we could potentially find up to $2^{2^{|V|}}$ many tangles. If we consider sets of three, then there can be at most $2a^{-1}|V|$ many $\mathcal{P}_\Psi$-tangles for any Ψ. Also, the tangle-tree-duality theorem (Diestel and Oum, 2021), which gives a witnessing dual, treelike structure for the absence of a tangle of a fixed set of separation, is richer when we consider sets of three instead of just pairs: If we consider pairs, this treelike structure of the separations system can only take the shape of a path, which is very restrictive. When considering sets of three, the nodes in this structure tree can have degrees of up to three; that is, they may branch. Lastly, increasing the size of the considered sets to more than three, increases computational complexity without introducing new mathematical behavior.

II. Algorithmic Details

We introduced our basic algorithmic framework in Section 3. In the following, we give details on the tangle algorithm and one specific post-processing. We additionally give insight into some algorithmic questions arising when applying tangles in practice and how these influence algorithmic decisions.

II.1 Details on Tangle Algorithm

In Algorithm 1, we present the main loop for the algorithm. Supposed we are given an agreement parameter a and a family of cuts $\mathcal{P} = \{P_i = \{A_i, A_i^{\complement}\}\}_i$ each of which has cost $\Psi_i \in \mathbb{R}$. We order $\mathcal{P}$ according to their cost Ψ_i, and then we iteratively try to add all the cuts. We terminate when either we cannot add a cut consistently or when we run out of cuts.

Consistency check: For each tangle τ that we have identified as non-maximal, we try to add both orientations of a new cut P to τ. Each orientation produces a potential new tangle τ'. We now need to check if τ' is consistent. We can check the consistency of τ' by checking the consistency of the *core* of τ', that is the set of most restricting bipartitions: the set of all the inclusion minimal sets in τ'. This is sufficient since if we have A, A', B, C such that $A' \subseteq A$, then

$$\left|A' \cap B \cap C\right| \geqslant a \implies |A \cap B \cap C| \geqslant a \tag{8}$$

Therefore, if a tangle's core is consistent, so is the tangle. If τ' is consistent, we add it as a left or right child in the tree. Which side depends on the orientation that created τ', left for A and right for $A^{\complement}$. Subsequently, we marked that it was possible to extend τ. If neither orientation can be added, we cannot extend τ and return.

In Algorithm 3 we show how to add a new orientation A to a tangle τ. To do so, we first add A to the specification of τ, then update the core of τ if necessary.

Algorithm 3: add orientation to tangle

```
Data: Node τ and new orientation A
Result: Child node τ_new
τ_new = {A} ∪ τ;
for C in core(τ) do
    if C ⊆ A then
        core(τ_new) = core(τ) ;
        return τ_new
    else if A ⊂ C then
        remove C from core(τ);
    end
end
core(τ_new) = {A} ∪ core(τ) ;
return τ_new
```

We postprocess the tree as described in Section 3.4. The final output of the post-processing approach is the attributed tree $(T^*, (p_t)_t)$, which is a "soft" version of a dendrogram, pseu-

Algorithm 4: post-processing the tangle search tree

Data: Tangle search tree T, weighting function $h : \mathbb{R} \to \mathbb{R}$, prune depth $\rho \in \mathbb{N}_0$
Result: Condensed tangle search tree T^*

`/* delete every node that has less than two children */`
for *node in* T **do**
  if *internal node has less than two children* **then**
    delete node from T ;
  end
end
`/* prune branches that are shorter than` ρ `*/`
while *not done* **do**
  for *leaf in* T **do**
    if *original distance to parent is shorter than* ρ **then**
      remove leaf;
      remove or contract parent if necessary ; `// every node needs to have exactly two children`
    end
  end
end
`/* bottom up propagate tangle information to get set` $\mathcal{P}$ `for each tangle` τ `*/`
while *root is not reached* **do**
  if *all children of a node* τ *orient a cut* P *the same* **then**
    track this information for the parent node ;
  else
    add P to $\mathcal{P}(\tau)$;
  end
end
compute $p_\tau^{(\text{right})}(v)$ according to Eq. 2 for every v and every τ ;
`/* derive probabilities of belonging to tangle` τ `by summing up the probabilities along the path from root to` τ `*/`
return $(T^*, (p_\tau)_\tau)$

docode is given in Algorithm 4. Since every node attribute $p_t \in [0,1]^n$ consists of probabilities for every object, the tree can be visualized with heatmaps as done in Figure 2.

II.2 Translating theory to practice

For transparency we want to mention discrepancies between the theoretical object and tangles as algorithmic pipeline. There are two main changes when moving from the theoretical object to practise.

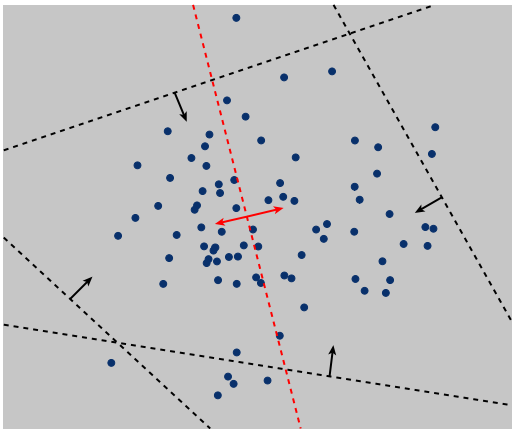


Figure 18: Gap problem. This figure depicts the intuition behind a large gap (here between the cheap bipartitions in black and the expensive bipartition in red). The red bipartition is orientable in both directions and would result in two tangles implying two clusters.

II.2.1 The gap problem

In theory, we consider all possible cuts for the set of initial cuts, and the agreement parameter is chosen small, only to assure tangles point to *something*. Let us consider a complete graph in which tangles are supposed to detect the existence of only one cluster (the whole dataset).

We then have the following intuition for tangles on $\mathcal{P}_\Psi$ (with suitable a): For small Ψ, all considered cuts are unbalanced, and the consistency condition forces every cut to orient towards the larger side. As Ψ increases, so does the balance of the cuts. At some point, $\mathcal{P}_\Psi$ starts to include cuts $P = (A, A^\complement)$ with a balance of $|A| = a$ and $\left|A^\complement\right| = n - a$ points that can be oriented both ways. However, the consistency condition prevents an orientation towards the smaller side A because $\mathcal{P}_\Psi$ also contains cuts that subdivide A. Eventually, even an orientation towards the larger side will become inconsistent. The procedure stops, and the tangle search tree is a path, as desired. However, in practice, it is infeasible to consider all possible cuts. Instead, some procedure generates a small subset of initial cuts $\mathcal{P}$, which is supposed to contain all relevant cuts for the cluster structure. One particular problem arises when there is a large **gap in the cost** of the cuts, indicating the next cut through more dense regions. The extreme case of random cuts illustrates this: Most random cuts split a dataset roughly in half. Although they cannot contain information about the cluster structure, we can orient the first random cut in the sorted list of cuts both ways for reasonable a. The second random cut roughly halves both sides of the first cut, yielding four subsets of about $n/4$ data points, which are consistent orientations for $a < n/4$. The critical difference between considering all possible cuts and just a subset is the presence or absence of other cuts: although both sets of cuts contain these high-cost random cuts, the former contains additional cuts that prevent them from being oriented consistently. When we consider all possible cuts, we rely on the fact that previous cuts (and the consistency condition) force specific orientations. In practice, we wish to derive an algorithm that re-

turns tangles pointing towards the same structure as the ones given all possible cuts.

Fine tuning parameters is an undesired solution. We could avoid problems caused by the gap problem by tuning the parameters a and Ψ. Setting Ψ low enough will avoid using expensive cuts, and we will immediately discard random or 'bad' cuts. By tuning the agreement parameter a, we can indirectly influence the size of the clusters we can discover. Setting this parameter large enough enforces larger clusters and avoids splitting into several smaller ones. We can also stop searching the tangle search tree as soon as we discover the desired number of clusters which can also avoid subdividing large structures randomly. However, usually, we know little about our data; we might not know how small the smallest cluster is nor the number of clusters we are looking for. We also consider this as one of the strengths of tangles; they require few parameter choices, so in practice, we try to avoid options that require guessing the properties of the data.

Dealing with a gap by pruning. We figured that often in practice, these "spurious" consistent orientations, as shown in Figure 18, become inconsistent after only a few additional cuts. To give some intuition, consider a set of random bipartitions. We expect each bipartition to roughly split our data in half, and these random samples are likely to be maximally dissimilar. So for each new cut, we quickly reduce the number of points within the intersection of every set of three cuts to drop below the agreement parameter. This phenomenon is something we can easily detect. We implement and explain this method in Section 3.4. While this proved to work well in practice in some cases, for a very unbalanced cluster structure, we might still get long paths in the tangle search tree even for a set of random cuts, and it requires us to set a parameter, the pruning depth. Luckily, this parameter often is easy to choose after looking at the tangle search tree.

Sensible sampling to avoid a gap. Random sampling is a naive approach to generating the initial set of bipartitions. Especially considering the gap problem as described above, random cuts are not sufficient. Trivially the algorithm works well if we only consider cuts that already split the data well, and the aggregation using tangles then yields good results. We need to be more careful if we consider an initial set with bad cuts. Ideally, we want to sample a rich set of initial cuts that, in the end, get oriented the way they would be if we considered all the cuts in between. We want a set of initial cuts that covers the set of all possible bipartitions in a way such that significant gaps do not appear (or are very unlikely). In this case, we try to mimic the theoretical setting but reduce the number of bipartitions dramatically, making it computationally feasible, even fast. In Section 6.1, we introduce one naive approach to generate such an initial set and prove that tangles resemble clusters in the considered setting.

II.2.2 The practical cost function

In tangle theory, the cost function does not consider the balance of the sides. For the cut value in the graph setting, the cheapest cuts are often very unbalanced, only splitting individual objects from the whole set and trivially get oriented towards their larger side. The same thing happens in practice. This does not prevent tangles from detecting clusters

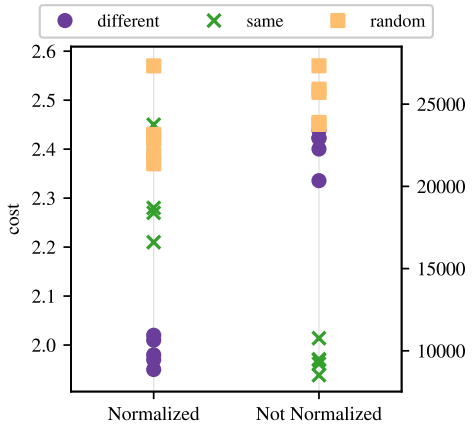


Figure 19: Order of the questions indicated by the costs with and without normalization. Normalizing the cost makes trivial questions (same answers) more expensive than informative questions (different answers) and starts with informative questions. The questionnaire consists of $k = 2$ mindsets and $m = 15$ questions. Five are answered differently by the mindsets, five are answered the same, and the remaining five are answered randomly by everyone. The marker indicates the question type.

but places an unnecessary computational burden on the tangle algorithm. Depending on the cost function, one natural countermeasure is to consider the balance of a cut $P = \{A, A^{\complement}\}$ by normalizing the cost with the factor $1/(|A|(n - |A|))$. Doing so makes the unbalanced cuts more expensive; tangles are built on informative cuts immediately.

For an illustrative example, consider the questionnaire model from Section 4.2 with $k = 2$ mindsets and small noise $p > 0$. Five questions are answered differently by the mindsets, five questions are answered the same, and another five questions are answered randomly by every person independently of the mindset. The questions with the same answer represent trivial, unbalanced cuts because they distinguish between persons that answer like their mindset (probability $1 - p$) and persons that answer differently (probability p). Only the questions with different answers are informative of the cluster structure. Figure 19 shows the cost of the questions with and without normalization. As expected, the unnormalized cost function places the trivial questions first. On the other hand, the normalized cost function increases their cost, such that the informative questions come first.

A stochastic block model gives another simplified example: two equal-sized blocks with 50 vertices each, $p = 0.3$, and $q = 0.1$. Figure 20 plots the (normalized) cost of a cut against its quality as measured by the normalized mutual information with the ground truth blocks. Since the basic idea is to use the cost of a cut as a proxy for its quality in separating clusters, a monotonic relationship between the two would be ideal. Normalizing increases the monotonic relationship strongly as measured by Spearman's rank correlation coefficient ρ (Zwillinger and Kokoska, 1999). This measure is equal to the Pearson correlation between the rankings induced by the variables and ranges from -1 to +1, where the extremal values

indicate a perfect monotonic relationship. Normalizing the cost in Figure 20 changes this coefficient from $\rho = -0.25$ (left plot) to a significantly stronger correlation of $\rho = -0.90$ (right plot).

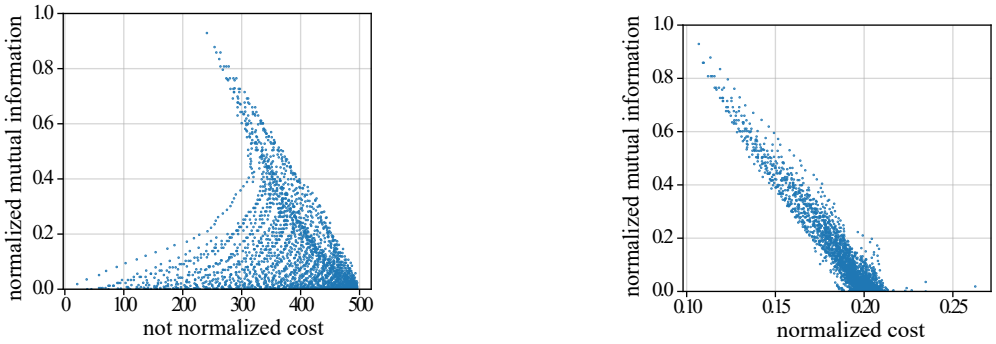


Figure 20: Normalized mutual information, of the cut given the ground truth, plotted against the (normalized) cost of a cut. Normalizing strongly increases the monotonic relationship between cost and quality of a cut.The dataset is a graph sampled from a stochastic block model with two blocks V_1 and V_2 with 50 vertices each, $p = 0.3$, and $q = 0.1$. For every $j, l \in \{0, \dots, 50\}$, one cut $P = \{A, A^C\}$ is randomly generated such that $|A \cap V_1| = j, |A \cap V_2| = l$. Each point in the plot corresponds to one cut.

III. Proofs

III.1 Binary Questionnaire

The proof of Theorem 2 is based on the following technical assumption, which excludes that in sampling the random mindsets μ_i we obtain a degenerate situation:

Assumption 1 (Mindsets are not degenerate) *If a tangle $\tau \in \{0,1\}^m$ satisfies that for all sets of three $x, y, z \leqslant m$ there exists a mindset μ_i such that tangle and mindset agree on all three: $\tau(x) = \mu_i(x)$ as well as $\tau(y) = \mu_i(y)$ and $\tau(z) = \mu_i(z)$, then the tangle τ is a single mindsets in the data, that is, $\tau = \mu_j$ for some j.*

Intuitively, Assumption 1 means we cannot trivially partition the ground truth into more than k tangles, for k ground truth mindsets. We now prove that for fixed k and growing m the probability of Assumption 1 being satisfied tends to 1:

Proposition 7 (Assumption 1 satisfied with high probability) *For $\beta = m/(k2^k)$, Assumption 1 is true with probability at least $1 - 2^{(1-\beta)k}$.*

Proof
We use a common bound for the coupon collectors problem (Motwani and Raghavan, 1995) to show that every partition of the mindsets is induced by one of the m questions with probability at least $1 - 2^{(1-\beta)k}$.
For a contradiction we suppose further, that there is some $\tau \in \{0,1\}^m$ with the property that for all questions $x, y, z \leqslant m$ there is some mindset μ_i such that $\tau(x) = \mu_i(x), \tau(y) = \mu_i(y)$ and $\tau(z) = \mu_i(z)$, but τ itself is not a mindset.
Let $x, y, z \leqslant m$ such that there are as few mindsets μ_i as possible with $\tau(x) = \mu_i(x), \tau(y) = \mu_i(y)$ and $\tau(z) = \mu_i(z)$ and suppose without loss of generality that μ_1 does so. Since every partition of the mindsets is induced by some question, there is a question x' such that $\mu_i(x') = \mu_i(x)$ for all $i \neq 1$ and $\mu_1(x') \neq \mu_1(x')$.
Now if $\tau(x') \neq \mu_1(x')$, then the set $\{x', y, z\}$ would contradict the fact that minimally many mindsets answered x, y, z the same way as τ.
Hence we may assume that $\tau(x') = \mu_1(x')$. Now, since $\tau \neq \mu_1$ there is some question $w \leqslant m$ with $\tau(w) \neq \mu_1(w)$. But then there is no mindset μ_i which answered the triple x, x', w in the same way as τ, which likewise contradicts the choice of x, y, z. ■

Proof of Theorem 3. We will prove the two directions of this statement separately, starting with showing that, with high probability, every mindset gives rise to a tangle:

Lemma 8 (Mindsets give tangles) *Let k, m be fixed and let $\alpha = a/n$ be the agreement parameter as fraction of n. If $p < (n - ka)/(3n)$, then the probability that there is one of the k mindsets which does not induce a tangle is bounded above by*

$$km \exp\left(-\frac{2n}{9k}(1 - k\alpha - 3p)^2\right),$$

which tends to 0 as n goes to ∞.

Intuition: For more questions and more mindsets it gets less likely that every ground truth mindset induces a tangle. The larger the number of questions m we consider the more likely that for one specific question, many (significantly more than $p \cdot n_i$) persons do not agree with their ground truth mindset. In this case the data might not represent the ground truth anymore, so we might not be able to recover it. If we consider a larger number of mindsets, we increase the number of possibilities to make an error. Also the number of persons per mindset decreases and thus for fixed α will get inconsistent if they become too small. It is intuitive that the probability of mindsets inducing tangles also decreases with increasing noise p. For a large number of persons n answering the questionnaire the probability of a statistical oilier decreases and so does the probability of mindsets not inducing tangles.

Proof If some mindset μ_i is not a tangle then it contains a subset of three questions with intersection at most a. Hence by the pigeon-hole principle for at least one of these three questions at most $(2n_i + a)/3$ of the n_i persons in V_i answered that question as μ_i does.
Each person in V_i answers a question as μ_i does with probability $(1 - p)$. We can apply Hoeffding's tail bound on the binomial distribution with success percentage $(1 - p)$. With X beeing the random variable counting the number of persons in V_i that answer a fixed question as μ_i does, we find that

$$\mathbb{P}\left[X \leqslant \frac{2n_i + a}{3}\right] \leqslant \exp\left(-2\frac{(\frac{2n_i+a}{3} - (1-p)n_i)^2}{n_i}\right) = \exp\left(-\frac{2n}{9k}(1 - k\alpha - 3p)^2\right),$$

Note that we used $a = n\alpha = kn_i\alpha$ here.
Since there are k mindsets and m questions, by the union bound we obtain that the probability of the event that there is some question for which few persons answer following their mindset is at most

$$km \exp\left(-\frac{2n}{9k}(1 - k\alpha - 3p)^2\right).$$

Consequently so is the probability that some mindset is not a tangle. ■

Lemma 9 (No degenerate tangles) *If $a = \alpha n$ and $p < a/n$, then, given Assumption 1, the probability that there is a tangle which is not a mindset is bounded above by $km \exp\left(-2(\alpha - p)^2 n/k\right)$.*

Intuition: Increasing the number m of questions or the number k of mindsets makes it more likely that there are tangles which do not completely agree with one of the mindsets. Like for Lemma 8, the larger the number m of questions or the number k of mindsets we consider, the more likely it is that for one specific question and one mindset, significantly more than $p \cdot n_i$ persons from that mindset do not agree with the ground truth of that mindset. This may then result in the existence of a tangle which agrees with one of the mindsets on all but this question, where the tangle chooses the opposite orientation, and thus corresponds to no existing mindset. For increasing k this is especially true, since increasing k with fixed n results in smaller values of n_i.

Proof By Assumption 1; for a tangle τ which does not correspond to a mindset there need to be a set of three x_1, x_2, x_3 such that for $y_i = \tau(x_1)$ we have that for $A_1 := A_{x_1}^{y_1}$, $A_2 := A_{x_2}^{y_2}$

and $A_3 \coloneqq A_{x_3}^{y_3}$ that $|A_1 \cap A_2 \cap A_3| \geqslant a$. However, no mindset μ can incorporate A_1, A_2, A_3, that is there is no μ with $\mu(x_i) = y_i \; \forall i \in \{1,2,3\}$.
So suppose that there are such A_1, A_2, A_3 with $|A_1 \cap A_2 \cap A_3| > a_Q$. Then there is, by the pigeon-hole principle, a mindset μ_i for which at least a/k many persons from V_i lie in $A_1 \cap A_2 \cap A_3$. If this mindset does not incorporate this triple, then it does not incorporate one of the questions, say $\mu_i(x_1) \neq y_1$.
We are going to compute the probability that there is some such A_x^y and a mindset μ_i where more than $n_i - a_Q/k$ many persons from V_i lie in A_x^y, but $\mu_i(x) \neq y$. Each person in V_i answers a question as μ_i does with probability $(1-p)$. Since it follows from $p \leqslant \alpha$ that $n_i - a/k \leqslant (1-p)n_i$, we can apply Hoeffding's tail bound on the binomial distribution with success probability $(1-p)$. With X beeing the random variable counting the number of persons in V_i that answer a fixed question as μ_i does, we find that

$$\mathbb{P}\left[X \leqslant n_i - \frac{a}{k}\right] \leqslant \exp\left(-2\frac{(n_i - \frac{a}{k} - (1-p)n_i)^2}{n_i}\right) = \exp\left(-\frac{2n}{k}(\alpha - p)^2\right),$$

Note that we used $a_Q = n\alpha_Q = kn_i\alpha_Q$ here.
Since there are k mindsets and m questions, by the union bound we obtain that the probability of the event that there is some such A_x^y and μ_i is at most

$$mk \exp\left(-\frac{2n}{k}(p - \alpha_Q)^2\right). \tag{9}$$

Consequently the probability that there exists a triple A_1, A_2, A_3 as above is also at most 9 since – as argued above – the A_1 in such a triple is such an A_x^y, witnessed by μ_i. ■

Proof of Theorem 2. It is easy to see that in the setting of Theorem 2 the requirements of the Lemmas 8 and 9 are satisfied. ■

III.2 Stochastic Block Model

We will show Theorem 3 in two stages. The first step is to calculate the maximum value of Ψ for which the two orientations of $\mathcal{P}_\Psi$ induced by V_1 and V_2 respectively are indeed tangles.
For this we shall compute the minimum cost of a *forbidden triple* in these orientations that is, a triple which violates a-consistency, that is a triple A, B, C of sides with the property that $|A \cap B \cap C| < a$. This we will do as follows: suppose that A, B, C is a forbidden triple in the orientation given by V_1. We will sequentially manipulate an existing such triple, while only reducing its cots, until we obtain a triple of a specific form, whose cost is easy to calculate.
The following lemma says that (for a certain range of parameters) the cost of cutting through both blocks is larger than cutting through only one block and moving the other block entirely to one side of the cut.

Lemma 10 (Not cutting through a block reduces the cost) *Let $P = \{A, A^\complement\}$ be a cut, and let $\alpha_i = |A \cap V_i|/|V_i|$ and $\beta_i = |A^\complement \cap V_i|/|V_i| = 1 - \alpha_i$. If $\alpha_2 \geqslant (1 - 2\alpha_1)q/p$, then*

$c(\{A \cup V_2, A^\complement \smallsetminus V_2\}) \leqslant c(P)$. *Similarly, if* $\beta_2 \geqslant (1-2\beta_1)q/p$, *then* $c(\{A \smallsetminus V_2, A^\complement \cup V_2\}) \leqslant c(P)$.

In words: Given a cut P which cuts through one of the blocks, say with a fraction α_2, if $\alpha_2 > (1-2\alpha_1)q/p$, then we can prove that the cut that results from moving all these $\alpha_2 n$ many points to the other side is cheaper.

Proof For $\alpha_2 = (1-2\alpha_1)q/p$ we have by (6), that

$$\begin{aligned} c(P) =& \frac{n^2}{4}\left(p(\alpha_1 - \alpha_1^2 + \alpha_2 - \alpha_2^2) + q(\alpha_1 + \alpha_2 - 2\alpha_1\alpha_2)\right) \\ =& \frac{n^2}{4}\left(p(\alpha_1 - \alpha_1^2) + q(1-\alpha_1)\right) = c(\{A \cup V_2, A^\complement \smallsetminus V_2\}) \end{aligned}$$

Now if we consider (6), for fixed n, p, q, α_1 as a real-valued function f mapping α_2 to $c(\{A, A^\complement\})$, then this is a quadratic function with a unique maximum. Thus by $f(1) = f((1-2\alpha_1)q/p)$ we have $f(x) \geqslant f(1)$ for every $(1-2\alpha_1)q/p \leqslant x \leqslant 1$, meaning for $\alpha_2 \geqslant (1-2\alpha_1)q/p$ we have that $c(\{A \cup V_2, A^\complement \smallsetminus V_2\}) \leqslant c(P)$.
The argument for $\beta_2 \geqslant (1-2\beta_1)q/p$ is analogous. ■

We will now compute the lowest possible cost of a forbidden triple A, B, C of which B and C both completely contain V_2 but A does not. We will later use Lemma 10 to see that the minimum cost forbidden triples are of this form.

Lemma 11 (Bound on the costs of a forbidden triple not cutting through a block) *For* $q/p \leqslant 1/2$, *if* A, B, C *are such that they each contain more than half of* V_1, *their intersection has size* $\leqslant a$, *and* V_2 *is a subset of* $A^\complement$, B, *and* C, *then the minimum possible value of* $\max\{c(\{A, A^\complement\}), c(\{B, B^\complement\}), c(\{C, C^\complement\})\}$ *is*

$$\frac{n^2}{4}p\left(\frac{1}{3}(1+r-x)(1+r) - (\frac{1+r-x}{3})^2\right), \text{ where } r = \frac{q}{p},\ x = \frac{2a}{n}.$$

with some such triple attaining this bound (up to discretization).

Proof We first calculate the cost $\{A, A^\complement\}$ would have if $|A^\complement \cap V_1| = |V_1|/2$.

$$c(\{A, A^\complement\}) = \frac{n^2}{4}\left(p(1 - 1^2 + \frac{1}{2} - \frac{1}{2}^2) + q(1 + \frac{1}{2} - 2\cdot 1\cdot\frac{1}{2})\right) = \frac{n^2}{4}p\left(\frac{1}{4} + \frac{r}{2}\right).$$

We observe for later that

$$\left(\frac{1}{3}(1+r-x)(1+r) - (\frac{1}{3}(1+r-x))^2\right) \leqslant \left(\frac{2}{9}(1+r)^2\right) \leqslant \frac{1}{4} + \frac{r}{2} \qquad \text{(C)}$$

for $0 < r \leqslant 1/2$. Moreover, by the calculations in the proof of Lemma 10, for fixed p, q, n and $\alpha_1 = 0$ the quadratic function given by (6) has its maximum at

$$\frac{1 + \frac{(1-2\alpha_1)q}{p}}{2} > \frac{1}{2}.$$

Thus this function is monotone in the interval between 0 and 1/2. Therefore, if A, B, C would be chosen such that $\max\{c(\{A, A^\complement\}), c(\{B, B^\complement\}), c(\{C, C^\complement\})\}$ is smaller than $n^2p\left(\alpha(1+r)-\alpha^2\right)/4$, both $|B^\complement \cap V_1|/|V_1|$ and $|C^\complement \cap V_1|/|V_1|$ need to be smaller than α. Therefore, in order for A, B, C to be a forbidden triple, $|A^\complement \cap V_1|/|V_1|$ needs to be larger than $(\alpha - r)|V_1|$. Since the function given by (6) is quadratic, and, by (C), its value at $1/2$ is larger than its value at $(\alpha - r)$, this would imply that the cost of $\{A, A^\complement\}$ is larger than $n^2p\left(\alpha(1+r)-\alpha^2\right)/4$, contradicting the assumption. ■

Proof of Theorem 3. We first show that τ^1_Ψ, and similarly τ^2_Ψ are indeed tangles. For this, suppose for a contradiction that $A, B, C \in \tau^1_\Psi$ such that $|A \cap B \cap C| < a$ and such that the maximum over the cost of the three is as small as possible. Then, by pigeon hole principle one of A, B, C must contain at least $(|V_2| - a)/3$ vertices from V_2.
By Lemma 10, we may now suppose without loss of generality, that$A^\complement \supseteq V_2$, say. Again by Lemma 10, we may additionally suppose that $B^\complement \cap V_2 = \varnothing$, $C^\complement \cap V_2 = \varnothing$. Thus by Lemma 11, the maximal cost of A, B, C is minimized by

$$\frac{n^2}{4}p\left(\frac{1}{3}(1+r-x)(1+r) - (\frac{1+r-x}{3})^2\right).$$

The argument for τ^2_Ψ is analogous.
Moreover, since the cost of the cut $\{V_1, V_2\}$ is $qn^2/4 < \Psi$, τ^1_Ψ and τ^2_Ψ are distinct as $V_1 \in \tau^1_\Psi$ and $V_2 \in \tau^2_\Psi$.
It remains to show that τ^1_Ψ and τ^2_Ψ are the only tangles. So suppose for a contradiction that there is an orientation τ of $\mathcal{P}_\Psi$ which is a tangle, but $\tau \notin \{\tau^1_\Psi, \tau^2_\Psi\}$. Then there is some $A \in \tau$ such that $|A^\complement \cap V_1| \geqslant |A \cap V_1|$. Suppose that $|A \cap V_1|$ is as small as possible; we will show that $|A \cap V_1| = 0$.
If $A \cap V_2 \neq \varnothing$, it is easy to see that the costs of the cut $\{A^\complement \cup V_2, A \smallsetminus V_2\}$ are at most the cost of the cut $\{A^\complement, A\}$. If τ contains $A \smallsetminus V_2$ we could chose the cut $\{A^\complement \cup V_2, A \smallsetminus V_2\}$ instead of $\{A^\complement, A\}$, so suppose that τ contains $A^\complement \cup V_2$.
As the costs of the cut $\{A \cap V_2, A^\complement \cup V_1\}$ are again at most the costs of $\{A, A^\complement\}$, our tangle needs to contain either $A \cap V_2$ or $A^\complement \cup V_1$. However the latter is not possible as $(A^\complement \cup V_1) \cap (A^\complement \cup V_2) \cap A = \varnothing$ contradicting the consistency of τ. Thus $A \cap V_2 \in \tau$ which already verifies that τ contains a set disjoint from V_1.
On the other hand, if $A \cap V_2 = \varnothing$ and there is is some $v \in A \cap V_1$ we can consider the cut $\{A^\complement \cup \{v\}, A \smallsetminus \{v\}\}$. Since $p \geqslant 2q$ the cost of this cut is, as $A \subseteq V_1$, strictly lower than that of $\{A, A^\complement\}$, hence τ contains an orientation of this cut. Since τ is a tangle with $a \geqslant 2$, and $A \cap (A^\complement \cup \{v\}) = \{v\}$, τ cannot contain $A^\complement \cup \{v\}$ and must therefore contain $A \smallsetminus \{v\}$. However, $A \smallsetminus \{v\}$ now contradicts the choice of A.
Thus τ contains an A disjoint from V_1, and similarly, as $\tau \neq \tau^2_\Psi$, the tangle τ contains an B disjoint from V_2. But since $A \cap B = \varnothing$ this contradicts the assertion that τ is a tangle. ■

To prove Theorem 4 we introduce the concept of locally minimal cuts. A cut $\{A, A^\complement\} \in \mathcal{P}$ is a *local minimum* if moving any single $v \in V$ to the other side does not decrease the cost. In generality for tangles for $a \geqslant 2$ every cut which is of minimum cost such that a given pair of tangles disagrees on it is a local minimum given the cost function.
However, all these local minimum cuts need to respect the blocks.

Lemma 12 (Local minimum cuts do not cut through the blocks) *Independently of the choice of parameters, as long as $p > 0$, every local minimum cut respects the blocks, that is if $\{A, A^\complement\}$ is a local minimum cut, then for $i = 1, 2$ either $V_i \subseteq A$ or $V_i \subseteq A^\complement$.*

Proof Suppose that $\{A, A^\complement\}$ is a local minimum cut but that, without loss of generality, both $V_1 \cap A$ and $V_1 \cap A^\complement$ are non-empty. Pick some arbitrary $v \in V_1 \cap A$ and $v' \in V_1 \cap A^\complement$. Since $\{A, A^\complement\}$ is locally minimal, the cut $\{A \setminus \{v\}, A^\complement \cup \{v\}\}$ has at least the cost of $\{A, A^\complement\}$, hence, as $w(v, v) = w(v', v') = 0$, we have

$$\sum_{x \in A} w(v, x) \geqslant \sum_{x \in A^\complement} w(v, x) \quad \text{and, similarly,} \quad \sum_{x \in A} w(v', x) \leqslant \sum_{x \in A^\complement} w(v', x)\,.$$

However, since v, v' both lie in V_1, we have that $w(v, x) = w(v', x)$ for all $x \neq v, v'$, thus:

$$\sum_{x \in A} w(v, x) \geqslant \sum_{x \in A^\complement} w(v, x) = \sum_{x \in A^\complement} w(v', x) + w(v, v') \geqslant \sum_{x \in A} w(v', x) + w(v, v') \geqslant \sum_{x \in A} w(v, x) + 2w(v, v')$$

Which is a contradiction as, $w(v, v') = p > 0$. ■

Proof of Theorem 4. Suppose there exist two tangles. Then there exists a lowest cost cut on which they disagree. This cut is a local minimum since $a \geqslant 2$. By Lemma 12 the only local minimum cuts are $\{\varnothing, V\}$ and $\{V_1, V_2\}$. So the cut in question must be $\{V_1, V_2\}$. As this cut has cost $qn^2/4$, our two tangles have to be $\mathcal{P}_\Psi$-tangles for some $\Psi \geqslant qn^2/4$.
Let τ be the tangle with $V_1 \in \tau$. Pick any two disjoint $X_1, X_2 \subseteq V_1$ with $|X_1| = |X_2| = \lfloor |V_1|/2 \rfloor$. We have that

$$c(\{X_1, X_1^\complement\}) = c(\{X_2, X_2^\complement\}) = p \left\lfloor \frac{n}{4} \right\rfloor \left\lceil \frac{n}{4} \right\rceil + q \left\lfloor \frac{n}{4} \right\rfloor \frac{n}{2} \leqslant p\frac{n^2}{16} + q\frac{n^2}{8} \leqslant q\frac{n^2}{4}.$$

So τ contains one of each X_1 or $X_1^\complement$ and X_2 or $X_2^\complement$. As $|V_1 \cap X_1^\complement \cap X_2^\complement| \leqslant 1 < a$, the tangle τ cannot contain both $X_1^\complement$ and $X_2^\complement$.
Without loss of generality we may assume that $X_1 \in \tau$ and that X_1 is of minimum size such that $X_1 \in \tau$. Now for any $x \in X_1$ the cut $\{X_1 \setminus \{x\}, (X_1 \setminus \{x\})^\complement\}$ has lower cost than $\{X_1, X_1^\complement\}$. Thus by the minimal choice of X_1 the tangle τ contains $(X_1 \setminus \{x\})^\complement$, but $\left|(X_1 \setminus \{x\})^\complement \cap X_1\right| = |\{x\}| = 1 < a$, which contradicts our assumption that τ is a tangle. ■

III.2.1 Identifying tangles from random cuts is hard

Note that all our results for the stochastic block model rely on the fact that $\mathcal{P}_\Psi$ contains *all* cuts of G up to cost Ψ rather than just a sample of those cuts. In practice $\mathcal{P}_\Psi$ might consist of many cuts, and so one usually would try to perform some sampling strategy to obtain a 'sensible' subset of these cuts. The following result shows that sampling from $\mathcal{P}_\Psi$ **uniformly at random is not a useful sampling strategy** for this purpose, as one would still be required to draw a sample of size exponential in n.
Observe that by Theorem 3 the blocks define $\mathcal{P}_\Psi$-tangles for *any* collection $\mathcal{P}$ of cuts. Thus, in order for the two blocks to define distinct tangles it suffices for the sampled set of cuts

to contain a single cut which *distinguishes* them – that is one cut $\{A, A^{\complement}\}$ such that more than half of V_1 lies in A and more than half of V_2 lies in $A^{\complement}$. Unfortunately, the number of these *good* cuts is exponentially small compared to the number of cuts up to any given cost Ψ:

Theorem 13 (Number of silly cuts) *Let p, q be fixed and let Ψ, a be chosen dependent on n such that the orientations induced by the blocks are distinct $\mathcal{P}_\Psi$-tangles for $\mathcal{P}_\Psi$ the set of all cuts up to cost Ψ. Then, asymptotically, the number of cuts not distinguishing these tangles is exponentially larger than the number of cuts distinguishing those tangles, for n going to infinity.*

This theorem implies that, if sampling cuts from $\mathcal{P}_\Psi$ uniformly at random, one would need to sample exponentially many of the cuts in order for the two blocks to define distinct tangles, since our sample needs to include one cut for this which distinguishes the blocks.

Proof of Theorem 13. We can construct all good partitions $\{A, A^{\complement}\}$ as follows: Starting with the partition $\{V_1, V_2\}$, we pick any subsets $G_1 \subseteq V_1$ of size $g_1|V_1| < |V_1|/2$ and $G_2 \subseteq V_2$ of size $g_2|V_2| < |V_2|/2$ and let $\{A, A^{\complement}\} = \{(V_1 \setminus G_1) \cup G_2\,, (V_2 \setminus G_2) \cup G_1\}$. We observe that the cost of $\{A, A^{\complement}\}$ depends only on g_1 and g_2. For fixed g_1 and g_2 there are exactly $\binom{|V_1|}{g_1 n} \cdot \binom{|V_2|}{g_2 n}$ many distinct cuts realizing this g_1 and g_2. In this case we say that the good cut $\{A, A^{\complement}\}$ *corresponds to* (g_1, g_2).
Similarly, we can construct all bad partitions $\{C, C^{\complement}\}$ as follows: Starting with the partition $\{\varnothing, V_1 \cup V_2\}$, we pick subsets $B_1 \subseteq V_1$ of size $b_1|V_1| < |V_1|/2$ and $B_2 \subseteq V_2$ of size $b_2|V_2| < |V_2|/2$ and let $\{C, C^{\complement}\} = \{B_1 \cup B_2, (V_1 \cup V_2) \setminus (B_1 \cup B_2)\}$. We observe that again, the cost of $\{C, C^{\complement}\}$ depends only on b_1 and b_2. Moreover for fixed b_1 and b_2 there are exactly $\binom{|V_1|}{b_1 n} \cdot \binom{|V_2|}{b_2 n}$ many distinct cuts realizing this b_1 and b_2. In this case we say that the bad cut $\{C, C^{\complement}\}$ *corresponds to* (b_1, b_2).
Therefore if we can show that for $b_1 = g_1 < 1/2$ and $b_2 = g_2 < 1/2$ the cost of $\{A, A^{\complement}\}$ as constructed above is always as least as big as the cost of $\{C, C^{\complement}\}$, this would imply that there are at least as many bad partitions as good ones, since we can construct an injective function from the set of good into the set of bad partitions of cost $< k$. The cost of a bad cut for b_1, b_2 is

$$\frac{n^2}{4}\left(p(b_1 - b_1^2 + b_2 - b_2^2) + q(b_1 + b_2 - 2b_1b_2)\right)$$

and the cost of a good cut for g_1 and g_2 is

$$\begin{aligned}&\frac{n^2}{4}\left(p(g_1 - g_1^2 + (1-g_2) - (1-g_2)^2) + q(g_1 + (1-g_2) - 2g_1(1-g_2))\right)\\ =&\frac{n^2}{4}\left(p(g_1 - g_1^2 + g_2 - g_2^2) + q(1 - g_1 - g_2 + 2g_1g_2)\right)\,.\end{aligned}$$

However, for $b_1, b_2 < 1/2$ we have that

$$2b_1 + 2b_2 - 4b_1b_2 < 1$$

and thus

$$b_1 + b_2 - 2b_1b_2 < 1 - b_1 - b_2 + 2b_1b_2\,,$$

meaning for $g_1 = b_1$ and $g_2 = b_2$ the cost of $\{C, C^\complement\}$ is indeed at most the cost of $\{A, A^\complement\}$. Therefore up to any given cost there are at least as many bad cuts as there are good ones. Moreover: For $\delta = 1 + q/p$ we can show that, under the condition that the cost of a good cut corresponding to (g_1, g_2) is at most $p\left(2\left(1+\frac{q}{p}\right)^2/9\right)$ (for larger Ψ, τ_k^1 is not a tangle by Lemma 11), then also the cost of a bad cut corresponding to $(\delta g_1, \delta g_2)$ is at most the cost of a good cut corresponding to (g_1, g_2).
Thus in this case we have, given a fixed Ψ and the set A_Ψ of all pairs (g_1, g_2) for which the cost of a good cut corresponding to (g_1, g_2) is at most Ψ, that there are exactly

$$\sum_{\substack{(i,j) \\ (\frac{i}{|V_1|}, \frac{j}{|V_2|}) \in A_\Psi}} \binom{|V_1|}{i} \cdot \binom{|V_2|}{j}$$

good cuts. However, there are at least

$$\sum_{\substack{(i,j) \\ (\frac{i}{\delta|V_2|}, \frac{j}{\delta|V_2|}) \in A_\Psi}} \binom{|V_1|}{i} \cdot \binom{|V_2|}{j}$$

bad cuts. Using these numbers we can see that for fixed $\delta > 1$ the number of bad cuts grows exponentially faster in the number of nodes than then number of good cuts, as n goes to infinity. ■

III.3 Feature based data

The following computations are in expectation, which we incorporate by assuming two things. First, we assume that the cost function behaves like the density function of the marginal distributions. Concretely, we assume the following:

Assumption 2 *The cost function c is such that if $(A_{j,i}, A_{j,i}^\complement)$ and $(A_{k,l}, A_{k,l}^\complement)$ are two axis-parallel bipartitions obtained from Algorithm 2, then $c(\{A_{j,i}, A_{j,i}^\complement\}) \leqslant c(\{A_{k,l}, A_{k,l}^\complement\})$ if and only if the density at $x_{j,i}$ of the marginal distribution on the j-axis is smaller than the density at $x_{k,l}$ of the marginal distribution on the k-axis.*

Secondly, we assume that the intersection of any three sides of bipartitions contains the fraction of the points that is expected from the density functions, that is, we assume the following:

Assumption 3 *Given three bipartitions $\{A, A^\complement\}, \{B, B^\complement\}, \{C, C^\complement\}$ in $\mathcal{P} = \bigcup_j \mathcal{P}_j$, where $A = \{v \in V \mid v_i < a\}$, $B = \{v \in V \mid v_j < b\}$, $C = \{v \in V \mid v_k < c\}$ for some $i, j, k \in \{1, \ldots, d\}$. Then the intersection of three sides of the bipartitions contains as many points as we would expect, that is, as large as the integral of the density over the intersection of the induced half-spaces.*

This means that we can use the quantiles of normal distributions in our arguments. In particular, we will use that for $\mathcal{N}(0,\sigma^2)$ less than 16% of the points are below $-\sigma$.
Let us say that a cut $\{A_{j,i}, A^{\complement}_{j,i}\}$ in $\mathcal{P}_j$ is a *local minimum* if and only if

$$c(\{A_{j,i-1}, A^{\complement}_{j,i-1}\}) > c(\{A_{j,i}, A^{\complement}_{j,i}\}), \qquad c(\{A_{j,i}, A^{\complement}_{j,i}\}) < c(\{A_{j,i+1}, A^{\complement}_{j,i+1}\}).$$

If we are given Assumption 2, we immediately obtain the following from the fact that the marginal densities of a Gaussian have exactly one local mimimum:

Observation 14 *If Assumption 2 holds, then there is, in every dimension, at most one separation which is a local minimum.*

We prove Theorem 5 by showing that, for the smallest possible order for which we find a local minimum, this local minimum is oriented differently by μ and ν. We show further, that this order is small enough such that the orientations induced by μ and ν cannot contain a triple of bipartitions with too small intersection.

Lemma 15 *If τ, τ' are distinct tangles, then there is a local minimum which is oriented differently by τ and τ'*

Proof Let $\{A_{j,i}, A^{\complement}_{j,i}\}$ be a cut of minimal possible cost distinguishing τ and τ', say $A_{j,i} \in \tau$ and $A^{\complement}_{j,i} \in \tau'$. If $\{A_{j,i}, A^{\complement}_{j,i}\}$ is not a local minimum, say because $\{A_{j,i-1}, A^{\complement}_{j,i-1}\}$ has lower cost (the other case is analogous), then both τ and τ' would need to orient $\{A_{j,i-1}, A^{\complement}_{j,i-1}\}$ similarly, and thus, since one of the two contains $A^{\complement}_{j,i}$, they would both need to contain $A^{\complement}_{j,i-1}$ by our consistency condition. However, since $\left|A_{j,i} \cap A^{\complement}_{j,i-1}\right| < a$ this contradicts the consistency of the tangle τ. ■

Lemma 16 *There exists a local minimum in $\mathcal{P}_j$ if and only if $|\mu_j - \nu_j| > 2\sigma$. Moreover if $(A_{j,i}, A^{\complement}_{j,i})$ is a local minimum, then either $\nu_j < x_{j,i} < \mu_j$, or $\mu_j < x_{j,i} < \nu_j$.*

Proof There is a local minimum in $\mathcal{P}_j$ if and only if the density function has a local minimum by Assumption 2. This is the case precisely if $|\mu_j - \nu_j| > 2\sigma$, see Helguerro (1904); Schilling et al. (2002). The second part is then also immediate, since the density function is monotone on the intervals $(-\infty, \mu_j)$ and (ν_j, ∞). ■

Proof of Theorem 5. We consider the set of all cuts in $\bigcup \mathcal{P}_j$ up to and including the cost of $\{A_{j,i}, A^{\complement}_{j,i}\}$, that is, $\{A_{j,i}, A^{\complement}_{j,i}\}$ is the only local minimum that we consider and it is also the most expensive cut that we consider.
Since the j-axis is the only axis on which we consider a local minimum, each cut which we are considering except $(A_{j,i}, A^{\complement}_{j,i})$ needs to have μ and ν on the same side, which we call the 'middle'. We define τ_μ by orienting $\{A_{j,i}, A^{\complement}_{j,i}\}$ as $A_{j,i}$, and all other cuts towards this middle.
Let us now show that this orientation is consistent; a tangle. Observe that, by Assumption 3, each side in τ_μ contains at least $n/2$ points. Further if we consider some three sides of cuts in

τ_μ which are along the same axis, then at most two of them are relevant to the intersection, since one of the sides will always be a superset of one of the others. So, considering three sides of any cuts in τ_μ, we need to consider just two cases:
Case I: All cuts are along distinct axes. Each of the three sides is expected to contain at least $n/2$ points, as observed. As the distributions along the distinct axes are independent, we thus have that the intersection of the three contains at least $n/8 > a$ points by Assumption 3.
Case II: Two cuts $\{A_{k,l}, A^\complement_{k,l}\}, \{A_{k,l'}, A^\complement_{k,l'}\}$ are along the same axis, the third is along another axis. We first determine the size of the intersection of the cuts along the same axis k. We claim that one of the two is chosen at distance more than σ from μ_k. Indeed, if one of the two cuts equals $\{A_{j,i}, A^\complement_{j,i}\}$ then this is true by the assumption of this theorem. Otherwise we claim that for one of the two cuts the point $x_{k,l}$ or $x_{k',l}$ has lower distance from ν_k, respectively $\nu_{k'}$, than from μ_k, respectively $\mu_{k'}$. Indeed, if for both of the two cuts the points $x_{k,l}$ and $x_{k',l}$ would have larger distance from ν_k and $\nu_{k'}$ than from μ_k and $\mu_{k'}$, respectively, then the size of the intersection of the sides of the two cuts contained in τ_μ would be equal to the size of the smaller of the two respective sides, hence we do not need to consider this triplet of cuts. Therefore, indeed we have for one of the two cuts, say for $\{A_{k,l}, A^\complement_{k,l}\}$, that $x_{k,l}$ has smaller distance from ν_k than from μ_k. Now if $x_{k,l}$ would have distance less than σ from μ_k, it would also need to have distance less than σ from ν_k, but this implies that this cut has higher costs than $\{A_{j,i}, A^\complement_{j,i}\}$, by Assumption 2, as $x_{j,i}$ has distance at least σ from both, μ_j and ν_j.
Thus, indeed one of the two cuts, say $\{A_{k,l}, A^\complement_{k,l}\}$, satisfies $|x_{k,l} - \mu_k| > \sigma$.
By Assumption 3, in expectation, at least 84% of the points belonging to μ – that is at least $0.42n$ points – lie on the same side of $\{A_{k,l}, A^\complement_{k,l}\}$ as τ_μ chooses. At least half of the points belonging to μ – that is at least $0.25n$ points – lie on the chosen side of the other cut along the same axis, $\{A_{k,l'}, A^\complement_{k,l'}\}$. Thus in their intersection there are expected to be at least $0.17n$ points.
The third cut is along an independent axis and, by the same argument as in Case I, at most halves this number. So there are already at least $0.085n > n/12 > a$ points in the intersection, alone from the points belonging to μ. ■

To prove Theorem 6 we will use the following simplification: Given a sampled cut $\{A_{j,i}, A^\complement_{j,i}\}$ there exists a whole interval in $\mathbb{R}$ in which we can chose an $x_{j,i}$ such that $A_{j,i} = \{v \in V \mid v_j < x_{j,i}\}$. To simplify the arguments, we will assume that our Assumptions 2 and 3 hold, regardless of how we have chosen $x_{j,i}$ for our given cut. This gives us the technical possibility to consider the cut $\{B_{j,x}, B^\complement_{j,x}\}$ sampled at $x \in \mathbb{R}$, that is, $B_{j,x} := \{v \in V \mid v_j < x\}$, for any $x \in \mathbb{R}$ and any dimension j. The result of this technicality is, that we do not have to take into account that, while we know that this cut is contained in $\mathcal{P}_j$, we may have put it into that set for a slightly different x. This simplifies the arguments a lot. We are aware, that as a formal statement this is an unrealistic assumption. However as the number of points tends to ∞, we will converge to this assumption, and thus we expect that the consequence of this theorem still holds.

Proof of Theorem 6. Suppose there is a tangle τ which points neither to μ nor to ν. Let us suppose that j is chosen such that $|\mu_j - \nu_j|$ is maximal. The general strategy of this proof is as follows: We will show that there are two local minima cuts in τ, one along the j-axis and one along another axis, such that one of the two points away from μ,

and the other points away from ν. The cost of these local minima cuts will allow us to find a set of three cuts which τ needs to orient a certain way, but which together violate the consistency condition on τ, due to the quantiles of the gaussian distribution. This will result in a contradiction to the assumption that τ is a tangle.
So let us first deal with the task of finding these local minima cuts.
As τ does not point towards μ, there exist a cut $\{B_{i,x}, B^{\complement}_{i,x}\}$ whose orientation in τ points away from μ, say $B^{\complement}_{i,x} \in \tau$ but $\mu_i < x$. We want to show that in this case we may suppose that $\{B_{i,x}, B^{\complement}_{i,x}\}$ is a local minimum.
So suppose that we cannot choose $\{B_{i,x}, B^{\complement}_{i,x}\}$ as a local minimum. We claim that there is an $x' \geqslant x$ and some $\epsilon > 0$ such that $B^{\complement}_{i,x'} \in \tau$ and $\{B_{i,x'+\epsilon'}, B^{\complement}_{i,x'+\epsilon'}\}$ is not oriented by τ for any $\epsilon' < \epsilon$. If we do not find such an x' and ϵ, then τ would need to orient every $\{B_{i,x'}, B^{\complement}_{i,x'}\}$ for any $x' > x$. However, since $B^{\complement}_{i,x} \in \tau$ we can conclude by the consistency condition that also $B^{\complement}_{i,x'} \in \tau$ for all $x' > x$ where $|x' - x|$ is small enough. Applying this argument inductively, we can conclude that in fact $B^{\complement}_{i,x'} \in \tau$ for all $x' > x$. But, for large enough x' we have that $\left|B^{\complement}_{i,x'}\right| < a$ which is a contradiction to the fact that τ is a tangle.
Thus we may suppose without loss of generality that our point x with $B^{\complement}_{i,x} \in \tau$ but $\mu \in B_{i,x}$ is chosen such that there is some $\epsilon > 0$ with the property that the cut $\{B_{i,x+\epsilon'}, B^{\complement}_{i,x+\epsilon'}\}$ is not oriented by τ for any $\epsilon' < \epsilon$. This means that x was chosen as large as possible with the property that $\{B_{i,x}, B^{\complement}_{i,x}\}$ is still oriented differently by τ and μ.
Now this choice of x implies that $\mu_i < x \leqslant \nu_i$ as otherwise, if $\mu_i \leqslant \nu_i < x$, the cut $\{B_{i,x+\epsilon}, B^{\complement}_{i,x+\epsilon}\}$ would, for any $\epsilon > 0$, have lower costs than $\{B_{i,x}, B^{\complement}_{i,x}\}$, contradicting the choice of x. Now, if $|\mu_i - \nu_i| > 2\sigma$, for $x' = \mu_i + \frac{|\mu_i - \nu_i|}{2}$ the cut $\{B_{i,x'}, B^{\complement}_{i,x'}\}$ is a local minimum and it is easy to see that τ contains $B^{\complement}_{i,x'}$ and thus this is a local minimum cut of the type that we assumed does not exists. Hence we have that $|\mu_i - \nu_i| \leqslant 2\sigma$.
Thus the density function along the i-axis has a unique local maximum between μ_i and ν_i, and the cost of $\{B_{i,\mu_i}, B^{\complement}_{i,\mu_i}\}$ is less than the cost of $\{B_{i,x}, B^{\complement}_{i,x}\}$. Now for any $y \in \mathbb{R}$, the cut $\{B_{j,y}, B^{\complement}_{j,y}\}$ along the j-axis (recall that j was chosen such that $|\mu_j - \nu_j|$ is maximal) has lower cost than $\{B_{i,\mu_i}, B^{\complement}_{i,\mu_i}\}$ as the cost of such a cut is maximal if y equals either μ_j or ν_j and in that case we still have that $|y - \nu_j| > 2\sigma > |\mu_i - \nu_i|$ or $|y - \mu_j| > 2\sigma > |\mu_i - \nu_i|$. However, this implies that τ needs to orient $\{B_{j,y}, B^{\complement}_{j,y}\}$ for every $y \in \mathbb{R}$ which is not possible without violating the consistency condition. Hence the assumption that we do not have a local minimum cut in τ pointing away from μ results in a contradiction to the fact that τ is a tangle.
So, there indeed needs to be a local minimum cut in τ which points away from μ. Similarly, we find a local minimum cut in τ which points away from ν. As j was chosen such that $|\mu_j - \nu_j|$ is maximal, τ in particular needs to orient the locally minimal cut $\{B_{j,l}, B^{\complement}_{j,l}\}$ in dimension j, since this cut is the local minimum cut of lowest possible cost. So let us suppose wlog. that τ orients this cut the same way as μ, that is, τ contains $B_{j,l}$ and $\mu_j < l < \nu_j$.
Further let $\{B_{i,x}, B^{\complement}_{i,x}\}$ be a local minimum cut that is oriented differently by τ and μ, so that $B^{\complement}_{i,x} \in \tau$ but $\mu_i < x$.
Since $\{B_{i,x}, B^{\complement}_{i,x}\}$ is a local minimum, we know that $|x - \mu_i| > \sigma$, as $|x - \mu_i| = |x - \nu_i|$ and $|\mu_i - \nu_i| > 2\sigma$ by the fact that there is a local minimum cut along the i-axis. Moreover, we can consider the cut $\{B_{j,z}, B^{\complement}_{j,z}\}$ corresponding to the point $z := \mu_j - |x - \mu_i|$ in dimension

j, which, by Assumption 2, is also oriented by τ since the density along the j-axis at z is less than the density along the i-axis at x. Since τ is a tangle, it needs to be the case that $B^{\complement}_{j,z}$ is contained in τ: If $B_{j,z} \in \tau$ this contradicts consistency, as τ, by Assumption 2, also orients all the cuts $\{B_{j,y}, B^{\complement}_{j,y}\}$ for $y \leqslant z$.

Furthermore let us consider the point $z' := \mu_j + |x - \mu_i|$. As $|x - \mu_i| < \frac{|\nu_j - \mu_j|}{2}$ we have that $|z' - \mu_j| = |x - \mu_i|$ and $|z' - \nu_j| \geqslant |x - \nu_i|$, and thus Assumption 2 again ensures that the cut $\{B_{j,z'}, B^{\complement}_{j,z'}\}$ in dimension j corresponding to z' has lower cost than $\{B_{i,x}, B^{\complement}_{i,x}\}$ and is thus oriented by τ. Since $B_{j,l} \in \tau$ it needs to be the case that $B_{j,z'} \in \tau$ as otherwise, τ would by Assumption 2 also need to orient all the cuts $\{B_{j,y}, B^{\complement}_{j,y}\}$ for $z' \leqslant y \leqslant l$ which would then contradict consistency.

We can now use Assumption 3 to calculate how many points are contained in $B^{\complement}_{j,z} \cap B_{j,z'} \cap B^{\complement}_{i,x}$. Let us first consider the points from μ. Let us denote the fraction of points from μ contained in $B^{\complement}_{i,x}$ as p, so $p = \left|V_\mu \cap B^{\complement}_{i,x}\right| \frac{2}{n}$. As $|x - \mu_i| > \sigma$, we have that $p < 0.16$.

By the choice of z we have that $B^{\complement}_{j,z}$ contains $(1-p)\frac{n}{2}$ points from μ and similarly, $B_{j,z'}$ contains a fraction of $(1-p)$ of the points from μ, namely $(1-p)\frac{n}{2}$ points.

So in total, by Assumption 3, the set $B^{\complement}_{j,z} \cap B_{j,z'} \cap B^{\complement}_{i,x}$ contains $p(1-p)^2\frac{n}{2}$ points from μ.

For the points from ν we observe that $B^{\complement}_{i,x}$ contains, by symmetry and the choice of p exactly $(1-p)\frac{n}{2}$ points from ν. $B_{j,z'}$ contains fewer points from ν than $B_{j,l}$ and $B_{j,l}$ contains $q \leqslant \frac{1}{2} \cdot \left(1 + \operatorname{erf}\left(\frac{-|\mu_j - \nu_j|}{(2\sqrt{2\sigma^2})}\right)\right)$ of the points from ν. So, again using Assumption 3, we can bound the total number of points contained in $B^{\complement}_{j,z} \cap B_{j,z'} \cap B^{\complement}_{i,x}$ by $\frac{n}{2}(p(1-p)^2 + q(1-p))$. As $p \leqslant 0.16$ and $q < p$, this is maximized for $p = 0.16$, where we obtain a value of about $n \cdot (0.42q + 0.056)$, hence if $a > n \cdot (0.42q + 0.056)$, τ cannot be a tangle, as claimed. ■

Questions in the Narcissistic Personality Inventory Dataset

	statement A	statement B
1	I have a natural talent for influencing people.	I am not good at influencing people.
2	Modesty doesn't become me.	I am essentially a modest person.
3	I would do almost anything on a dare.	I tend to be a fairly cautious person.
4	When people compliment me I sometimes get embarrassed.	I know that I am good because everybody keeps telling me so.
5	The thought of ruling the world frightens the hell out of me.	If I ruled the world it would be a better place.
6	I can usually talk my way out of anything.	I try to accept the consequences of my behavior.
7	I prefer to blend in with the crowd.	I like to be the center of attention.
8	I will be a success.	I am not too concerned about success.
9	I am no better or worse than most people.	I think I am a special person.
10	I am not sure if I would make a good leader.	I see myself as a good leader.
11	I am assertive.	I wish I were more assertive.
12	I like to have authority over other people.	I don't mind following orders.
13	I find it easy to manipulate people.	I don't like it when I find myself manipulating people.
14	I insist upon getting the respect that is due me.	I usually get the respect that I deserve.
15	I don't particularly like to show off my body.	I like to show off my body.
16	I can read people like a book.	People are sometimes hard to understand.
17	If I feel competent I am willing to take responsibility for making decisions.	I like to take responsibility for making decisions.
18	I just want to be reasonably happy.	I want to amount to something in the eyes of the world.
19	My body is nothing special.	I like to look at my body.
20	I try not to be a show off.	I will usually show off if I get the chance.

	statement A	statement B
21	I always know what I am doing.	Sometimes I am not sure of what I am doing.
22	I sometimes depend on people to get things done.	I rarely depend on anyone else to get things done.
23	Sometimes I tell good stories.	Everybody likes to hear my stories.
24	I expect a great deal from other people.	I like to do things for other people.
25	I will never be satisfied until I get all that I deserve.	I take my satisfactions as they come.
26	Compliments embarrass me.	I like to be complimented.
27	I have a strong will to power.	Power for its own sake doesn't interest me.
28	I don't care about new fads and fashions.	I like to start new fads and fashions.
29	I like to look at myself in the mirror.	I am not particularly interested in looking at myself in the mirror.
30	I really like to be the center of attention.	It makes me uncomfortable to be the center of attention.
31	I can live my life in any way I want to.	People can't always live their lives in terms of what they want.
32	Being an authority doesn't mean that much to me.	People always seem to recognize my authority.
33	I would prefer to be a leader.	It makes little difference to me whether I am a leader or not.
34	I am going to be a great person.	I hope I am going to be successful.
35	People sometimes believe what I tell them.	I can make anybody believe anything I want them to.
36	I am a born leader.	Leadership is a quality that takes a long time to develop.
37	I wish somebody would someday write my biography.	I don't like people to pry into my life for any reason.
38	I get upset when people don't notice how I look when I go out in public.	I don't mind blending into the crowd when I go out in public.
39	I am more capable than other people.	There is a lot that I can learn from other people.
40	I am much like everybody else.	I am an extraordinary person.

Published in Transactions on Machine Learning Research (09/2023)

Relating graph auto-encoders to linear models

Solveig Klepper *solveig.klepper@uni-tuebingen.de*
Department of Computer Science and Tübingen AI Center
University of Tübingen

Ulrike von Luxburg *ulrike.luxburg@uni-tuebingen.de*
Department of Computer Science and Tübingen AI Center
University of Tübingen

Reviewed on OpenReview: *https://openreview.net/forum?id=Y1eYplvxrE*

Abstract

Graph auto-encoders are widely used to construct graph representations in Euclidean vector spaces. However, it has already been pointed out empirically that linear models on many tasks can outperform graph auto-encoders. In our work, we prove that the solution space induced by graph auto-encoders is a subset of the solution space of a linear map. This demonstrates that linear embedding models have at least the representational power of graph auto-encoders based on graph convolutional networks. So why are we still using nonlinear graph auto-encoders? One reason could be that actively restricting the linear solution space might introduce an inductive bias that helps improve learning and generalization. While many researchers believe that the nonlinearity of the encoder is the critical ingredient towards this end, we instead identify the node features of the graph as a more powerful inductive bias. We give theoretical insights by introducing a corresponding bias in a linear model and analyzing the change in the solution space. Our experiments are aligned with other empirical work on this question and show that the linear encoder can outperform the nonlinear encoder when using feature information.

1 Introduction

Many real-world data sets have a natural representation in terms of a graph that illustrates relationships or interactions between entities (nodes) within the data. Extracting and summarizing information from these graphs is the key problem in graph learning, and representation learning is an important pillar in this work. The goal is to represent/encode/embed a graph in a low-dimensional space to apply machine learning algorithms to solve a final task.
In recent years neural networks and other approaches that automate learning have been employed as a powerful alternative to leverage the structure and properties of graphs (Tang et al., 2015; Grover & Leskovec, 2016; Ribeiro et al., 2017; Dong et al., 2017; Hamilton et al., 2017a; Chamberlain et al., 2017; Cao et al., 2016; Wang et al., 2016; Chang et al., 2015). For example, Hamilton et al. (2017b) review key advancements in this area of research, including graph convolutional networks (GCNs). Graph auto-encoders (GAEs), first introduced in Kipf & Welling (2016b), are based on a graph convolutional network (GCN) architecture. They have been heavily used and further refined for representation learning during the past couple of years (see Li et al. (2021); Pan et al. (2018); Vaibhav et al. (2019); Davidson et al. (2018), to name a few). Recently, Salha et al. (2020) discovered that the original GAE architecture, which has been used as a basis for several enhancements, can be outperformed by simple linear models on an exhaustive set of tasks. The equivalence or even superiority of (almost) linear models has also been empirically shown for standard GCN architectures for link prediction and community detection (Wu et al., 2019). The question arises as to why we still use and improve these models when simpler models with comparable performance exist. What is the theoretical relation between linear and nonlinear encoders, and does the empirical work provide a reason to replace

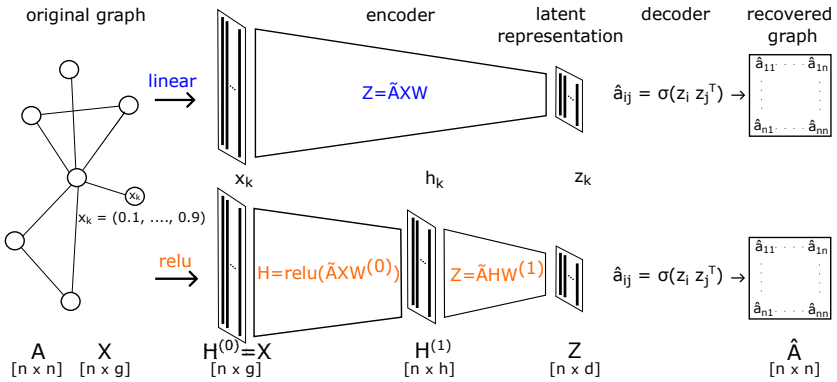


Figure 1: The architecture of a linear auto-encoder and a relu auto-encoder. The only difference is the encoder function: The linear encoder is a simple linear map, whereas the relu encoder is a two-layer GCN network with relu activation in the first layer and linear output activation. In both cases, the input is the graph (A, X). $H^{(1)}$ is the hidden layer of the GCN. A row in the matrix Z gives the latent representation of the respective row/node in the input matrix X. In both cases the decoder is the inner product, and the target is to recover the adjacency matrix A.

graph convolutional networks with linear equivalents? In this paper, we contribute to understanding the underlying inductive bias of the graph auto-encoder model.

Contributions: (1) We draw a theoretical connection between linear and non-linear encoders and prove that, under mild assumptions, for any function $f(A, X)$ on a graph, there exists a linear encoder that can achieve the same training loss. We use this result to show that the representational power of relu encoders is, at most, as large as the one of linear encoders. (2) We investigate the bias that is relevant for good generalization and give theoretical insights into how features influence the solution space of a linear model. (3) We empirically find that the nonlinearity in the relu encoder is not the critical ingredient for generalization. Indeed we discover that if the graph features are taken into account appropriately, the linear model outperforms the relu model test data.

2 Preliminaries

We consider a connected, unweighted and undirected graph G with adjacency matrix $A \in \{0,1\}^{n\times n}$ and a feature matrix $X \in \mathbb{R}^{n\times g}$. The feature matrix X contains g-dimensional feature vectors that describe additional properties of each node of the graph. For example, if the graph consists of a social network, the features could describe additional properties of the individual persons, such as age and income. If we do not have any feature information for the nodes ("featureless graph"), then we set X as the $n \times n$ identity matrix. We always assume that all nodes are connected to themselves, that is, $a_{ii} = 1$ for all i. We call $D \in \mathbb{N}^{n\times n}$ the degree matrix with diagonal entries $d_{ii} = \sum_{j=1}^{n} A_{ij}$ and 0 everywhere else, and $\tilde{A} = D^{-1/2}AD^{-1/2}$ the symmetrically normalized adjacency matrix.

Graph convolutional networks. Inspired by convolutional neural networks, graph convolutional networks (GCNs, Kipf & Welling (2016a)) are one of the most widely used graph neural network architectures. The input consists of a graph (A, X). The graph convolution is applied to a matrix $\tilde{A}$ that encodes the adjacency structure of the graph. This matrix $\tilde{A}$, called diffusion matrix, is the symmetrically normalized adjacency matrix in our case but could be replaced by any matrix encoding the graph structure, for example, the graph Laplacian. In each layer of the network, a node in the graph updates its latent representation, aggregating feature information within its neighbourhood, and learns a low-dimensional representation of the graph and its features. More precisely, in layer l, each node update is some function of the diffusion of the weighted

features: $H^{(l+1)} = \psi(\tilde{A}H^{(l)}W^{(l)})$ with learnable weights $W^{(l)}$, (nonlinear) activation function ψ and $H^{(0)}$ as the feature matrix X. The output of the GCN for each node in the graph is a d-dimensional vector representing this node's embedding. This representation can be used for downstream tasks such as node or graph classification or regression.

Graph auto-encoders. Graph auto-encoders aim to learn a mapping of the graph input (A, X) to an embedding $Z \in \mathbb{R}^{n\times d}$. This embedding is optimized such that, given a decoder, we can recover the adjacency matrix A from the embedding. As visualized in Figure 1, the encoder consists of a two-layer graph convolutional network with no output activation and hidden layer with size $h \in \mathbb{N}$. The decoder is a simple dot product:

$$\underbrace{Z_{\text{relu}} = \text{GCN}(X, A) = \tilde{A}\ \text{relu}(\tilde{A}XW^{(0)})W^{(1)}}_{\text{encoder}}, \quad \underbrace{\hat{A}_{\text{relu}} = \sigma(Z_{\text{relu}}Z_{\text{relu}}^{\mathsf{T}})}_{\text{decoder}}. \tag{1}$$

Here $Z_{\text{relu}} \in \mathbb{R}^{n\times d}$ is the latent space representation with embedding dimension d, and the matrix $\hat{A}$ is the reconstructed adjacency matrix. Note that $\hat{A}$ is a real-valued matrix that aims to capture the likelihood of each edge in the graph. To obtain a binary adjacency matrix as a final result, we discretize by thresholding the values at 0.5. The loss function optimized by the auto-encoder is the cross-entropy between the target matrix A and the reconstructed adjacency matrix $\hat{A}$. It is minimized over the encoder's parameters $W^{(l)}$. The decoder is a deterministic function and is not learned from the data. For a graph with adjacency matrix A, latent space embedding Z, and with $\sigma(\cdot)$ denoting the sigmoid function, the loss is formalized as follows:

$$l_A(Z) = \sum_{ij=1}^{n} a_{ij}\log(\sigma(z_i z_j^{\mathsf{T}})) + (1 - a_{ij})(1 - \log(\sigma(z_i z_j^{\mathsf{T}}))). \tag{2}$$

Below we compare GAEs based on a relu encoder with a simplified model based on a linear encoder.

3 Relu encoders have at most the representational power of linear encoders

Graph auto-encoders are widely used for representation learning of graphs. However, it has also been observed that linear models outperform them on several tasks. In this section, we investigate this empirical observation from a theoretical point of view. We define a graph auto-encoder that uses a simple linear map as the encoder and prove that the derived model has larger representational power than a relu encoder. To this end, we introduce a linear encoder based on a linear map of the diffusion matrix $Z_{\text{lin}} = \tilde{A}W$, where Z_{lin} is the latent space representation. The loss function and the decoder remain as in the relu model. For a relu encoder, the dimension h of the hidden layer satisfies $g > h > d$ (for the linear encoder, the parameter h does not exist). We now define the solution spaces to help us describe the respective models' behavior. They will represent the embeddings that can be learned by relu and linear encoders, both with and without node features and more general functions on graphs:

$$\begin{aligned}
\mathcal{Z}_{\text{lin}} &= \{Z \in \mathbb{R}^{n\times d} | Z = \tilde{A}W \text{ for } W \in \mathbb{R}^{n\times d}\} \\
\mathcal{Z}_{\text{lin},X} &= \{Z \in \mathbb{R}^{n\times d} | Z = \tilde{A}XW \text{ for } W \in \mathbb{R}^{g\times d}\} \\
\mathcal{Z}_{\text{relu}} &= \{Z \in \mathbb{R}^{n\times d} | Z = \tilde{A}\,\text{relu}(\tilde{A}W^{(0)})W^{(1)} \text{ for } W^{(0)} \in \mathbb{R}^{n\times h}, W^{(1)} \in \mathbb{R}^{h\times d}\} \\
\mathcal{Z}_{\text{relu},X} &= \{Z \in \mathbb{R}^{n\times d} | Z = \tilde{A}\,\text{relu}(\tilde{A}XW^{(0)})W^{(1)} \text{ for } W^{(0)} \in \mathbb{R}^{g\times h}, W^{(1)} \in \mathbb{R}^{h\times d}\} \\
\mathcal{Z}_{f} &= \{Z \in \mathbb{R}^{n\times d} | Z = f(\tilde{A}, X; \theta) \text{ for } \theta \in \Theta\}.
\end{aligned}$$

Here f can be any parameterized function on a graph $f : \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times g} \times \Theta \to \mathbb{R}^{n\times d}$ and Θ can be an arbitrary set of parameters. It is easy to see that $\mathcal{Z}_{\text{lin},X} \subseteq \mathcal{Z}_{\text{lin}}$. In words: when we compare two linear encoders, one with node features and one without node features, the one with node features has a more restricted solution space. The reason is that the feature matrix X is at most rank g and typically has a low rank compared to the diffusion matrix $\tilde{A}$, which restricts the image of $\tilde{A}X$. The same holds for the relu encoder: $\mathcal{Z}_{\text{relu},X} \subseteq \mathcal{Z}_{\text{relu}}$. The relation between the linear encoder and the relu encoder is less obvious. One might guess that the relu encoder can learn more mappings than a linear encoder. However, we now show that this is not the case: we will see that $\mathcal{Z}_{\text{lin}} \supseteq \mathcal{Z}_{\text{relu}}$, focusing on the featureless model. This relation implies that the linear encoder has a larger representational power than a relu encoder.

While this paper focuses on a specific architecture that has been introduced in previous work and is heavily used in practice, some of our insights extend to any encoder function f under mild assumptions on the input graph. Thus, we first state our result in a more general way. We make the following assumptions:

Assumption 1. The number of nodes n is larger than the feature dimension g.

Assumption 2. The rank of the diffusion matrix $\tilde{A}$ is larger than or equal to the embedding dimension d.

Assumption 3. The loss-function l is invariant to orthogonal transformations, that is, $l_A(Z) = l_A(RZ)$ for any orthogonal matrix R.

• Assumption 1 is typically fulfilled in practice. Vertices are typically described by a moderate number of features. Especially for larger graphs, this is a weak assumption.
• Assumption 2 is usually satisfied because the embedding dimension is a choice, and a graph auto-encoder aims to find a low-dimensional representation. We can also think about this assumption in the context of the informational content of our data. If the input matrix is of low rank, it holds little information; we will not gain information by embedding it into a higher dimensional space. Setting the embedding dimension d smaller than the rank of the input matrix $\text{rank}(\tilde{A})$ makes sense.
• Assumption 3 is fulfilled in the graph auto-encoder setting; the loss given in Equation 2 is based on pairwise distances given by the dot-product. This loss is invariant to orthogonal transformations. Generally, one could construct loss functions not invariant to orthogonal transformations. However, orthogonal transformations usually preserve pairwise metrics, and as a result, all loss functions based on a decoder incorporating pairwise distances will also be invariant to orthogonal transformations.

To prove our main theorem, we first observe that given Assumption 2, for any set P of n points in a d-dimensional subset of $\mathbb{R}^n$, we can find an orthogonal transformation that maps the span of A so that it contains all points in P.

Lemma 4. *For $P \in \mathbb{R}^{n\times d}, A \in \mathbb{R}^{n\times m}$ with $\text{rank}(P) \leqslant rank(A)$, there exists an orthogonal matrix R with $\text{span}(P) \subseteq \text{span}(RA)$.*

We can now prove that the linear model can outperform any other function f on the graph G during training.

Theorem 5 (Minimal Loss). *Consider a fixed graph $G = (A, X)$. Let $\tilde{A}$ be the diffusion matrix of G and let $Z \in \{Z \in \mathbb{R}^{n\times d} \mid Z = f(A, X; \theta)$ for $\in \Theta\}$ be any latent space representation of any graph function $f(A, X; \theta)$. Let l be a loss function and assume that Assumptions 1-3 are satisfied. Then the linear model can find a a latent space representation with loss at least as good as the relu encoder.*

Proof. With Lemma 4 it holds that for any $Z \in \mathbb{R}^{n\times d}$ and any matrix $\tilde{A} \in \mathbb{R}^{n\times m}$ with $\text{rank}(\tilde{A}) \geqslant \text{rank}(Z)$, there exists an orthogonal matrix $R \in \mathbb{R}^{n\times n}$ such that all points in latent space representation Z are in the image of $R\tilde{A}$:

$$\exists R \text{ orthogonal s.t. } \{R\tilde{A}W \mid W \in \mathbb{R}^{m\times d}\} \supseteq \{Z \mid Z = f(A, X; \theta) \in \mathbb{R}^{n\times d}\}.$$

This implies that there exists a weight matrix $W \in \mathbb{R}^{m\times d}$ with $l(Z) = l(R\tilde{A}W) = l(\tilde{A}W)$. In other words, the minimal loss over the set of possible solutions that is learnable by the linear model is smaller or equal to any loss over the solution space achievable by the function f:

$$\inf\{l(Z_f) \mid Z_f \in \mathcal{Z}_f\} \geqslant \inf\{l(Z_{\text{lin}}) \mid Z_{\text{lin}} \in \mathcal{Z}_{\text{lin}}\}$$

□

Theorem 5 considers a quite general statement. The graph auto-encoder architecture is one particular instance of a function f. Given the specific architecture of the auto-encoder we can show that the solution space of the relu-encoder is contained in the solution space of the linear model.

Corollary 6 (Representational power of GAEs). *For any (trained) graph auto-encoder in $\mathcal{Z}_{\text{relu}} \supseteq \mathcal{Z}_{\text{relu},X}$, there exists an equivalent, featureless linear encoder in $\mathcal{Z}_{\text{lin}}$ that can achieve the same training loss because we have $\mathcal{Z}_{\text{lin}} \supseteq \mathcal{Z}_{\text{relu}}$.*

We refer to Appendix A for the proof. Note that if we assume that $\text{rank}(AF) > d$, then the rank of AF is larger than the embedding dimension d. Theorem 5 and Corollary 6 hold for the solution space of the linear model, including features: $\mathcal{Z}_{\text{lin,X}}$. We discuss the influence of the features on the representational power of the linear model in Section 4.

Corollary 6 has a simple but strong implication: in principle, the (featureless) linear model is more powerful than the traditional GAE (with or without features). Here "more powerful" means that the linear model can achieve better training loss.

The issue of representational power. Note that we do not claim that

$$\{f : \mathbb{R}^{n\times g} \to \mathbb{R}^{n\times d} \mid f(A, X) \text{ is any nonlinear GNN}\} \subseteq \{f : \mathbb{R}^{n\times g} \to \mathbb{R}^{n\times d} \mid f(A, X) = A\, W \text{ for } W \in \mathbb{R}^{g\times d}\},$$

which would mean that the function class of linear models contains the function class of nonlinear models. However, in the setting of a graph neural network, we usually have a single graph as an input with a fixed number of nodes n. In this setting, the goal is to find an embedding of this fixed graph. Our result says that $\mathcal{Z}_f \subseteq \mathcal{Z}_{\text{lin}}$ and in particular $\mathcal{Z}_{\texttt{relu,X}} \subseteq \mathcal{Z}_{\texttt{relu}} \subseteq \mathcal{Z}_{\texttt{lin}}$. In words: The solution space of the nonlinear model is contained in the solution space of the linear one, showing that the representational power of the linear model is larger than the one of the nonlinear model.

Generalization properties. Note that Corollary 6 does not claim that the linear model without features outperforms the relu model. This corollary only considers the training loss. Thus we can not derive anything about the test performance or the solution that an optimization algorithm might find. So far, we are just consider training loss and the existence of a weight matrix W. From the above discussion, it is apparent that the linear encoder is less restricted, which means that it introduces a weaker inductive bias than the relu encoder. Note that this is independent of the fact that we used the relu activation function or the depth of the convolutional network. Regarding the test performance, the question is whether the restriction that the relu encoder puts on the solution space induces a "good" inductive bias that helps improve the model's test error. As we see below, we empirically find that when using features, a relu encoder does outperform the (featureless) linear encoder on test data. The relu activation and the features both restrict the solution space and introduce an inductive bias. Next, we investigate which of these two ingredients helps to restrict the representational power in a meaningful way.

4 The inductive bias of adding features

This section investigates how features influence the solution space of the linear model and the inductive bias that we introduce by adding features. While Corollary 6 shows the superior representational power of the linear model without features, in Section 5, we will observe improved test performance for models that do use feature information. Whenever the features are low rank compared to the diffusion matrix, they restrict the solution space and introduce an inductive bias that facilitates learning and can improve generalization if they hold helpful information. We observe this behaviour for both considered architectures, the relu and the linear encoder. Consequently, we conjecture that the presence of features, and not the relu nonlinearity, introduces the necessary bias to restrict the solution space in a meaningful way. However, some care is needed. As we will see, adding features can also harm performance if they contradict the structure of the target graph. In order to quantify the harm, we derive a measure for misalignment between graph structure and feature information. Based on this, we give theoretical insights about the restriction of the solution space when including features in the linear model.

Intuitively, we want to call a graph A and the corresponding features X "aligned" if they encode "similar" information. In our setting, this would be the case if A is "close to" XX^T: considering the features $X \in \mathbb{R}^{n\times g}$ as node embeddings, the matrix XX^T reconstructed adjacency matrix in a similar way as the dot product decoder. In other words, nodes should be close together in the feature space if and only if an edge connects them. Since the adjacency matrix is symmetric, a decomposition $A = YY^T$ for $Y \in \mathbb{C}^{n\times n}$ always exists. The error that we might introduce by adding features then comes from the fact that XX^T is a (low rank) approximation of $YY^\mathsf{T} = A$: Even for perfectly aligned features, the feature dimension g is typically much

smaller than n. Note that if we consider one graph and two different feature matrices with the same span, both features restrict the solution space in the same way and give rise to the same optimal embedding:

Proposition 7 (Features with the same span induce the same solution space). *Let $\tilde{A} \in \mathbb{R}^{n \times n}$ be the diffusion matrix of a fixed graph and $U, F \in \mathbb{R}^{n \times g}$ two different feature matrices for this graph. If $\text{span}(U) = \text{span}(F)$, then $\mathcal{Z}_{\text{lin},U} = \mathcal{Z}_{\text{lin},F}$.*

Intuitively, this means that if U and F span the same space, the corresponding models can, in principle, learn the same mappings and output the same embeddings.

Proof. We show that for every weight matrix W_U with $Z = \tilde{A}UW_U$, there exists a weight matrix W_F with $Z = \tilde{A}FW_F$. From $\text{span}(U) = \text{span}(F)$ it follows that $\text{span}(\tilde{A}U) = \text{span}(\tilde{A}F)$. We define corresponding linear maps $f_U, f_F : \mathbb{R}^d \to \mathbb{R}^n$ with $f_U(x) = \tilde{A}Ux$ and $f_F(x) = \tilde{A}Fx$. Since $\text{Im}(f_U) = \text{Im}(f_F)$ we know that every point $f_U(x) = y$ has at least one pre-image in f_F. Let $W_U = [w_{U_1}, ..., w_{U_d}]$ and $Z = [z_1, ..., z_d]$ be such that $z_i = f_U(w_{U_i})$. For every z_i there exists at least one w_{F_i} with $f_F(w_{F_i}) = z_i = f_U(w_{U_i})$. Since this holds for any matrix W_U we can conclude that $\mathcal{Z}_{\text{lin},U} = \mathcal{Z}_{\text{lin},F}$. □

We have seen that two different feature matrices with the same span restrict the solution space equivalently. Next, we derive a condition for the alignment between features X and graph structure $\tilde{A}$ that quantifies when features are potentially helpful. To do so, we investigate two settings. In the first setting, the features align with the graph structure and can introduce a valuable restriction of the solution space. In the second setting, the features contradict the graph structure and restrict the solution space in such a way that the optimal embedding can no longer be recovered. The intuition for helpful features corresponds with the one for good embeddings: If points are connected in the graph, they should have similar feature vectors. Consequently, if we were not to embed the nodes into a low-dimensional space but were simply interested in any latent representation of the graph, the features themselves would represent desirable embeddings.

Proposition 8 (Features cannot hurt if they perfectly align with the graph structure). *Let X be the feature matrix of the graph with diffusion matrix $\tilde{A}$ and let the embedding dimension coincide with the feature dimension g. If $\text{Im}(\tilde{A}X) = \text{Im}(X)$, then there exists a weight matrix W with $X = \tilde{A}XW$.*

This proposition shows that if $\text{Im}(\tilde{A}) = \text{Im}(\tilde{A}X)$, then the model can recover the features.

Proof. Since $\tilde{A}X$ and X span the same subspace, we can find a w_2 for any w_1 with $Xw_1 = \tilde{A}Xw_2$ and $w_1, w_2 \in \mathbb{R}^g$. We interpret every column in a matrix as an independent vector to derive that for any matrix W_1 we can find a matrix W_2 with $XW_1 = \tilde{A}XW_2$ and $W_1, W_2 \in \mathbb{R}^{g \times g}$. □

The proposition suggests that if the features encode the same structure as the graph's adjacency matrix, they do not negatively restrict the solution space. We now consider the opposite case: if the node relations encoded by their features contradict the adjacency structure of the graph, we can neither recover the features nor the adjacency matrix. In this setting, features restrict the function space in a harmful way, preventing the easiest and trivially optimal embedding. From these observations, we then derive a definition to measure the alignment between a graph's structure and node features.

Proposition 9 (Features can hurt if they do not align with the graph structure). *Let X be the graph's feature matrix, and assume that the embedding dimension coincides with the feature dimension, $d = g$. Let $\tilde{A}$ be the diffusion matrix and let $Y \in \mathbb{R}^{n \times d}$ be a matrix such that $\tilde{A} = YY^T$. Assume that the graph can be perfectly represented in g dimensions, that is, there exists a $Z \in \mathbb{R}^{n \times g}$ with $ZZ^T = \tilde{A}$. Let $\text{rank}(Y) = \text{rank}(\tilde{A}) = g$.*

1. *If $\text{Im}(\tilde{A}X) \neq \text{Im}(X)$, then there exists no weight matrix W with $X = \tilde{A}XW$.*
2. *If $\text{Im}(\tilde{A}X) \cap \text{Im}(X)^\perp \neq \varnothing$, then there exists no weight matrix W with $Y = \tilde{A}XW$.*

The intuition is that if the adjacency matrix and the node features define two different structures, then neither of the structures can be encoded by the linear model. The alignment of the structures is captured by the condition on the image of $\tilde{A}X$ and X.

Proof of Part 1. We prove the contraposition: If there exists a weight matrix W with $X = \tilde{A}XW$ then $\mathrm{Im}(\tilde{A}X) = \mathrm{Im}(X)$. Let $\tilde{A} \in \mathbb{R}^{n\times n}$ and let $X \in \mathbb{R}^{n\times g}$ be full rank. Let W be such that $X = \tilde{A}XW$. Since X is full rank, the rank of W is full and thus $\mathrm{Im}(X) = \mathrm{Im}(\tilde{A}XW) = \mathrm{Im}(\tilde{A}X)$. □

Proof of Part 2. For $Y = \tilde{A}XW = YY^{\mathsf{T}}XW$ to hold we need W to be the $g \times g$ inverse of (Y^TX). We prove that if $\mathrm{Im}(\tilde{A}X) \cap \mathrm{Im}(X)^{\perp} \neq \varnothing$, then $Y^{\mathsf{T}}X \in \mathbb{R}^{g\times g}$ is of rank smaller g and thus not invertible. We again prove the contraposition: If $X^{\mathsf{T}}Y$ is full rank then $\mathrm{Im}(\tilde{A}X) \cap \mathrm{Im}(X)^{\perp} = \{0\}$. Note that $Y^{\mathsf{T}}X$ is full rank by assumption and thus: $\mathrm{Im}(\tilde{A}X) = \mathrm{Im}(YY^{\mathsf{T}}X) = \mathrm{Im}(Y)$. Generally it holds that $\mathrm{Im}(X)^{\perp} = \ker(X^{\mathsf{T}})$. Let $w \in \mathrm{Im}(Y) \cap \ker(X^{\mathsf{T}})$. Since w is in $\mathrm{Im}(Y)$ there exists a $v \in \mathbb{R}^d$ with $w = Yv$. Since w in $\ker(X^{\mathsf{T}})$, it follows that $X^{\mathsf{T}}Yv = X^{\mathsf{T}}w = 0$. This implies $v = (X^{\mathsf{T}}Y)^{-1}0 = 0$ and thus $w = 0$ and thus $\mathrm{Im}(Y) \cap \ker(X^{\mathsf{T}}) = \{0\}$. □

Note that the dot product is invariant under orthogonal transformations; thus, for the features, every orthogonal transformation can be absorbed in the weight matrix. Proposition 9 also holds for orthogonally transformed features and embeddings. Thus given the dot-product as a decoder there is no other embedding $\tilde{Y} \neq Y$ that recovers the adjacency structure of the graph.

Based on all these insights, we propose a misalignment definition between a graph and its node features. Intuitively, we want to say that A and X are misaligned if $\tilde{A}X$ and X do not span the same subspace.

Definition 10 (Misalignment of graph and features). Let $\tilde{A}$ be the symmetrically normalized adjacency matrix and $\tilde{X}$ the normalized feature matrix such that each column has l_2-norm 1. We measure misalignment of a graph A and its features X using the distance $d_{\mathrm{algn}} : \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times g} \to \mathbb{R}$, where the arccos function is applied to every entry in the matrix:

$$d_{\mathrm{algn}}(A, X) \mapsto \mathrm{Tr}(\arccos(\tilde{A}\tilde{X}\tilde{X}^{\mathsf{T}})).$$

We got inspired to this measure by the geodesic distance on the Grassmannian manifold, which measures the principal angles of two subspaces AX and X. If d_{algn} is low, the alignment between A and X is high. It is easy to see that with this formulation, $d_{\mathrm{algn}}(A, X) = 0$ if $A = XX^T$. This is consistent with our intuition that the features and structure of the graph are perfectly aligned if they encode the same structure.

The generalization error of linear encoders is task dependent. Learning and generalization can only work if the learning architecture encodes the correct bias for the given task. We now consider two tasks to demonstrate the negative and positive impact of the bias that we introduce when using features.

In recent work, the most common task is **link prediction**. For this task we consider a graph given by its adjacency matrix $A \in \{0, 1\}^{n\times n}$ and a feature matrix $X \in \mathbb{R}^{n\times g}$. We construct a train graph $\bar{A}$ by deleting a set of edges from the graph, which we later, during test time, would like to recover. We train to minimize the loss $l(\bar{A}, X)$, while the test loss is given by $l(A, X)$. In our opinion, link prediction is a somewhat strange task when talking about generalization. We assume that the input graph is incomplete or perturbed. During training, we optimize the GAE for an adjacency matrix that we do not wish to recover during test time (because we also want to discover the omitted edges). However, this cannot be encoded in the loss of the auto-encoder (compare Eq. (2)). For the encoding to discover the desired embedding, the necessary bias to prevent the model from optimally fitting the training data must be somewhere else in the model. We claim that the features introduce this necessary bias, and we evaluate the link prediction task because it is so prominent in the literature.

We consider a second task, **node prediction**. For this task, the is graph given by its adjacency matrix $A \in \{0, 1\}^{n\times n}$ and the feature matrix $X \in \mathbb{R}^{n\times g}$. We construct a train graph $\bar{A}, \bar{X}$ by deleting a set of nodes from the graph A and the corresponding features from the feature matrix X. We train to optimize $l(\bar{A}, \bar{X})$. We then compute the test loss on the larger graph A, X: $l(A, X)$ with the goal of predicting the omitted nodes. The goal is to learn a mapping from the input graph to the latent space that generalizes to new, unseen vertices.

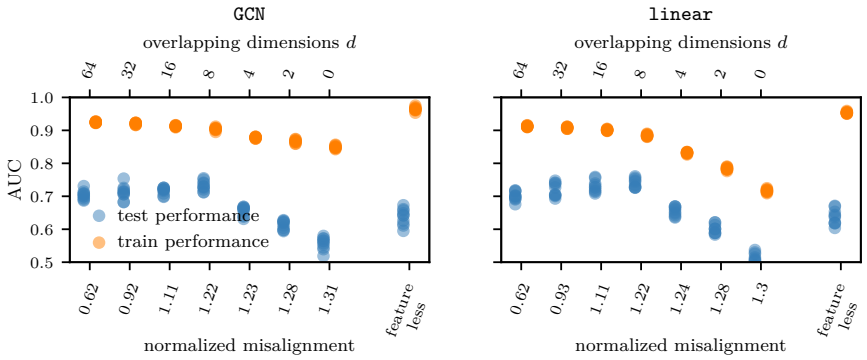


Figure 2: When features help. In the link prediction task, features help to regularize if they align with $\tilde{A}$ because they encode the target structure. Along the x-axis we plot the alignment between the graph and the features with two different scores: on the bottom, we plot misalignment as measured by d_{algn}, described in Definition 10 (low is good alignment, high is bad alignment). On the top, we show the alignment given by the number of overlapping dimensions of the two subspaces. The figures show train and test performance for the link prediction task on the synthetic dataset described in Section 5. We embed the points into eight dimensions. We plot values for ten independently sampled graphs. The left figure shows results for the architecture with the relu encoder and the right figure shows results for the linear model.

Intuitively it is clear that, depending on the task, the usefulness of the introduced bias might vary. We empirically evaluate the influence of features and compare the performance of the linear and the nonlinear encoder with and without the bias of features.

5 Empirical evaluation

In this section, we evaluate the influence of node features and the role of their alignment both for linear and relu encoders. We will see that the results support the theory that linear encoders outperform relu encoders.

Setting. We consider two models for empirical evaluation. The first is the relu encoder defined in Equation (1) based on the implementation of Kipf & Welling (2016b), and the second is the linear encoder. Since the relu encoder has two layers and thus two weight matrices, we realize the weights for the linear model accordingly: $Z_{\text{lin}} = \tilde{A}XW^{(0)}W^{(1)}$. This definition does not contradict the one from above; we can simply choose $W = W^{(0)}W^{(1)}$. Regarding the representational power of the model, this makes no difference. However, from an optimization perspective, it can influence the training (see Saxe et al. (2013)), and we aim to compare the models as fair as possible. We use the same objective function as Kipf & Welling (2016b), weighting the edges to tackle the sparsity of the graph and regularizing by their mean squared norm to prevent the point embeddings from diverging. We train using gradient descent and the Adam optimizer for 200 epochs. Due to sparsity, the accuracy of the recovered adjacency matrix is not very insightful. As is done in previous work, we measure the performance of the auto-encoders by the area under the receiver operating characteristic curve (AUC). To get a representative metric we use all present (positive) edges from the graph and uniformly sample the a number of what we call 'negative' edges, that is, edges that are not present in the graph's adjacency matrix. As a result, we only use some of the entries of the adjacency matrix for evaluation and get a balanced set of present and non present edges in the graph. To get representative results, we run every experiment 10 times. In real-world datasets, we randomize over the test set, drawing different edges or nodes each time. For the synthetic datasets, we draw ten independent graphs from the same generative model described below, then construct the test and train sets on these.

Datasets. We use the following generative model to get a **synthetic dataset** with aligned features. We draw $n = 1000$ data points from a standard Gaussian distribution in $g = 64$ dimensions. We normalize the

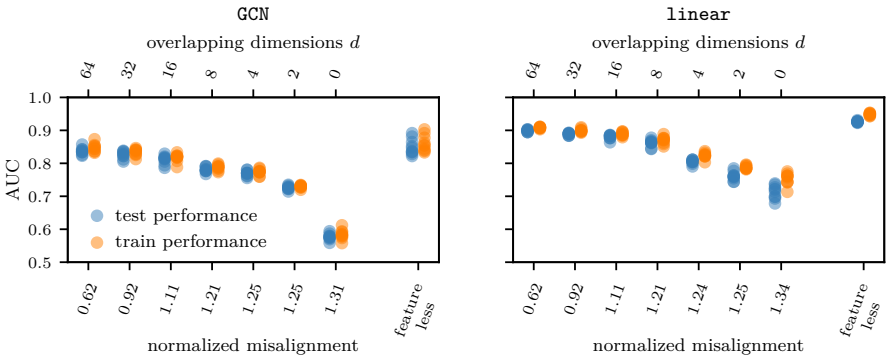


Figure 3: When features harm. Adding features can harm the node prediction task performance, preventing optimal graph fitting. The x-axis shows the alignment between the graph and the features with two different scores: we plot the misalignment as measured by d_{algn}, described in Definition 10 (low is good alignment, high is bad alignment). On the top, we show alignment given by the number of overlapping dimensions of the two subspaces. The figures show train and test performance for the node prediction task on the synthetic dataset described in Section 5. We embed the nodes into eight dimensions. We plot values for ten independently sampled graphs. The left figure shows results for the architecture with the relu-encoder and the right figure shows results for the linear model.

points to lie on the unit sphere and store their coordinates in a feature matrix $X \in \mathbb{R}^{n \times g}$. We then construct the graph by thresholding the matrix $XX^\intercal$ to get a 0-1-adjacency matrix of density between 0.01 and 0.02. With this construction, the feature matrix and the graph adjacency matrix are as aligned as possible, given the dimensional restriction. We generate data that simulates misalignment as follows. We aim to construct features that overlap in d dimensions with the graph's optimally aligned, true features. Recall that changing the span does not restrict the solution space, so we can replace the features with any other basis of the same space. The feature matrix X is spanned by $\{u_1, ..., u_g\}$. We construct a perturbed feature matrix $X_{\texttt{perturbed}}$ that has the same rank as X and spans the first d dimensions of the space; span$(u_1, ..., u_d)$, but is orthogonal to the remaining space; span$(u_{d+1}, ..., u_g)$. We do so by calculating the singular value decomposition $X = U\Sigma V^\intercal$, then choose the first d columns and the last $g - d$ columns from U. Let $U = [u_1, ..., u_n]$, we set $X_{perturbed} = [u_1, ..., u_d, u_{n-(g-d)}, ..., u_n]$. Note that the vectors $u_{n-(g-d)}, ..., u_n$ lie completely in the orthogonal complement of span$(u_1, ..., u_g)$, which means that the span of the optimal features X and the perturbed features $X_{\texttt{perturbed}}$ overlap in exactly d dimensions. We calculate the misalignment between the graph A and the perturbed features $X_{\texttt{perturbed}}$ as in Definition (10). By this construction, we do not fix the degree of rotation of the non-overlapping dimensions. As a result, the misalignment values for an overlap of a specified dimension d can vary slightly in different runs. However, as we normalize the values with regard to the dimensions, we still get insights for quantitative comparison. For example, we get a misalignment value d_{algn} of about 0.92 for an overlap of 32 dimensions on the synthetic dataset with 64 dimensions overall. This corresponds to an overlap of approximately 50% of the dimensions. We get a misalignment value d_{algn} of approximately 1.11 for an overlap of 16 dimensions, corresponding to an overlap of approximately 25% of the dimensions.

As **real world datasets** we consider three standard benchmarks: `Cora`, `Citeseer`, and `Pubmed`. Similar to previous work, we embed the nodes into 16 dimensions when considering the link prediction task. For the node prediction task, embedding into 16 dimension turns out to be too simple a task. We thus use 4 as the embedding dimension. For graph statistics of all used datasets and more details on the experimental setup, see Appendix B.

When features help for link prediction. In Figure 2, we plot the AUC performance of both encoder variants, relu and linear, both with and without features, on the link prediction task. We observe that

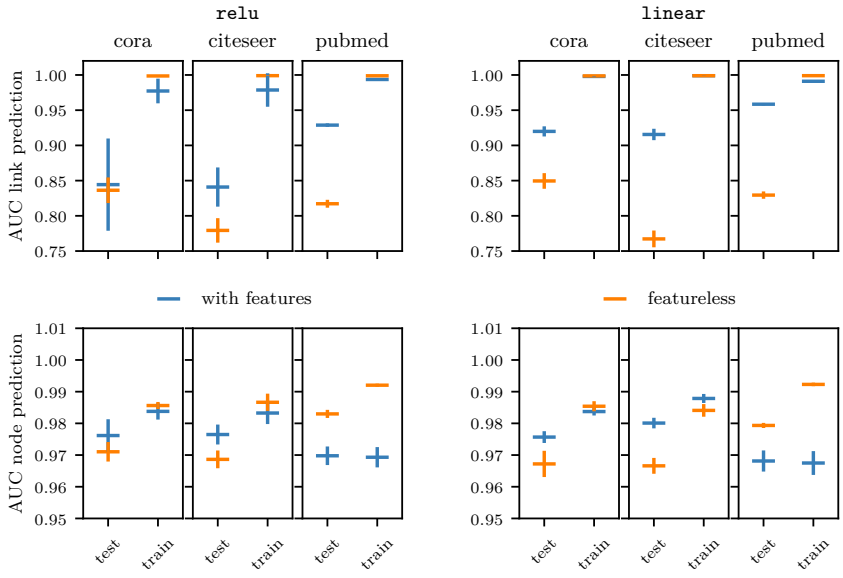


Figure 4: Train and test performance for the three real-world datasets `CoraCiteseer`and `Pubmed`. Vertical lines indicate standard deviation. We consider relu and linear encoder models with and without features. Top: Link prediction task; we embed points into 16 dimensions. Bottom: Node prediction task, points are embedded into four dimensions. The normalized misalignment scores are 0.67 for `Cora`, 0.69 for `Citeseer`and 0.86 for `Pubmed`.

both models' test performances usually decrease with increasing misalignment between the features and the graph. Adding features decreases the train performance, indicating that features restrict the solution space. Features may encode similarities in the feature space that are not present in the (incomplete) graph. This information restricts the solution space and can help to recover those missing structures. Moreover, the featureless model shows high train performance while the test performance is low. This behavior is expected in link prediction as optimal train performance implies sub-optimal test performance. In Figure 4, we observe the same behavior for the real-world datasets. The features add information about the target structure and improve the test performance. We suggest the following perspective: In the link prediction task, features act as a regularizer preventing the model from encoding the training adjacency matrix optimally. If they encode the desired graph structure, this regularization makes the model more robust to perturbations in the adjacency matrix, which is basically the task for link prediction. Here we assume the input graph is incomplete or perturbed, and we want to be robust against these perturbations and still recover the original graph. If the features do not encode the desired structure, including them can harm the performance. The relu encoder, where the features are passed through a weight multiplication and a relu activation, turns out to be more robust to false information, possibly learning to ignore (parts) of the features. The linear model can not compensate for erroneous feature information, which becomes visible in Figure 2, where, with more significant misalignment, the training error for the relu encoder drops slowly compared to the linear encoder. We can suspect slight overfitting in both settings when the overlapping dimension exceeds the embedding dimension. The effect is declining test performance, even for very aligned features. In this case, the introduced bias via the features might be too weak.

When features harm. In Figure 3, we visualize the performance of the models in the node prediction task. We observe similar behavior for the architectures using features as in the previous task. As expected, the performance decreases for an increasing distance of alignment d_{algn}. Interestingly, the test performance stays close to the train performance, which indicates that both models generalize well to unseen nodes and larger

graphs. Supporting our intuition, we observe that for both encoders, the featureless versions outperform the architectures using features. In the node prediction task, restricting the representational power by adding features harms the model's performance. Our experiments for the node prediction task show that featureless models can outperform the versions including features. Contrary to the link prediction task, in the node prediction setting, we assume that the given graph adjacency matrix already encodes the target structure of every node in the graph. Even if the features align optimally with the adjacency matrix, they do not hold additional information that is not already encoded in the adjacency matrix. In this setting, features still add a bias, but assuming that the given adjacency matrix is noiseless, they restrict the solution space in an undesired way.

In practice and real-world data, we usually do not know whether the given features encode the correct structure or if the graph is complete. So, not all of the intuition discussed above directly applies. We compute the normalized distance as given by d_{algn} and observe that the alignment is generally very good, with values of 0.6709 and 0.6940 for `Cora` and `Citeseer`, respectively. Misalignment is still good but slightly worse for `Pubmed` with a value of 0.8577, which indicates that almost 50% of the dimensions are misaligned. For the three benchmarks and the node prediction task, embedding into 16 dimensions already achieves nearly perfect recovery for all models. This reinforces our discussion about Assumption 2 being weak to no restriction. Usually, one would choose the smallest possible embedding dimension that shows promising results. See Appendix C for the performance when embedding into 16 dimensions. We embed the data into only four dimensions to get more insightful results. In Figure 4, we see that for the `Cora` and the `Citeseer` datasets, the models using features perform slightly better than those without features. This behavior could indicate noise in the input adjacency matrix, which the feature information regularizes. For the `Pubmed` dataset, where the features are slightly less aligned, not using the features seems to perform better. However, this difference is minimal. Compared to the synthetic dataset and `Pubmed`, the two networks from `Cora` and `Citeseer` have significantly higher feature dimensions in relation to the number of nodes possibly encoding redundancy. In summary, our findings indicate that the features encode mostly the correct structure, and adding them can improve optimization even for features with almost no additional information.

6 Conclusion

This work presents a theoretical perspective on the representational power and the inductive bias of graph auto-encoders. We consider a relu architecture and a linear architecture for the encoder. We prove that the linear encoder has greater representational power than nonlinear encoders. Theorem 5 also extends to more advanced models with similar encoder architecture and even to other nonlinear graph functions. Based on our experiments and empirical work in the literature, the nonlinear structure of the relu auto-encoder does not improve learning compared to the linear encoder. Our evaluations support the idea that the introduced bias from the nonlinearity is not the crucial ingredient to reducing the solution space in a meaningful way. On the other hand, the features can introduce a powerful inductive bias to both encoder architectures, improving their test performance. Whether features improve training and test performance heavily depends on the task we want to solve. For the two example tasks we consider in this paper, features help with link prediction but do not help node prediction. However, supporting the idea that linear encoders have larger representational power in training and can generalize, the linear encoder outperforms the nonlinear one in both tasks.

Acknowledgments

This work has been supported by the German Research Foundation through the Cluster of Excellence Machine Learning New Perspectives for Science" (EXC 2064/1 number 390727645) and the International Max Planck Research School for Intelligent Systems (IMPRS-IS).

References

Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. AAAI Conference on Artificial Intelligence, 2016.

Benjamin Paul Chamberlain, James Clough, and Marc Peter Deisenroth. Neural embeddings of graphs in hyperbolic space. *preprint arXiv:1705.10359*, 2017.

Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C. Aggarwal, and Thomas S. Huang. Heterogeneous network embedding via deep architectures. International conference on knowledge discovery and data mining (KDD), 2015.

Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. Hyperspherical variational auto-encoders. Conference on Uncertainty in Artificial Intelligence (UAI), 2018.

Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. Metapath2vec: Scalable representation learning for heterogeneous networks. International conference on knowledge discovery and data mining (KDD), 2017.

Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. International conference on Knowledge discovery and data mining (KDD), 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. Neural Information Processing Systems (NeurIPS), 2017a.

William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *preprint arXiv:1709.05584*, 2017b.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *preprint arXiv:1609.02907*, 2016a.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NeurIPS Workshop on Bayesian Deep Learning*, 2016b.

Xuelong Li, Hongyuan Zhang, and Rui Zhang. Adaptive graph auto-encoder for general data clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. International Joint Conference on Artificial Intelligence, 2018.

Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. Struc2vec: Learning node representations from structural identity. International conference on knowledge discovery and data mining (KDD), 2017.

Guillaume Salha, Romain Hennequin, and Michalis Vazirgiannis. Simple and effective graph autoencoders with one-hop linear models. European Conference, ECML PKDD, 2020.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. pp. 1067–1077. International conference on world wide web (WWW), 2015.

Vaibhav, Po-Yao Huang, and Robert Frederking. Rwr-gae: Random walk regularization for graph auto encoders. *arXiv preprint arXiv:1908.04003*, 2019.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. International conference on Knowledge discovery and data mining (KDD), 2016.

Felix Wu, Tianyi Zhang, Amauri H. Souza Jr., Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. International Conference on Machine Learning (ICML), 2019.

Published in Transactions on Machine Learning Research (09/2023)

A Proofs

In this Section, we provide the proofs of Lemma 4 and Corollary 6 and discuss the consequences.

Lemma 4 (Orthogonal Transformation). For $P \in \mathbb{R}^{n\times d}, A \in \mathbb{R}^{n\times m}$, $\text{rank}(P) \leqslant rank(A)$, there exists an orthogonal matrix R with $\text{span}(RA) \supseteq \text{span}(P)$.

Proof of Lemma 4. $\text{rank}(Z) \leqslant rank(A) = k$.
$\text{Im}(A)$ is spanned by an orthogonal normal basis $\{u_{i=1,\dots,d,\dots k}\}$
$\text{Im}(Z)$ is spanned by an orthogonal normal basis $\{v_{i=1,\dots,d}\}$.

$$\text{Let } U := \left(\begin{array}{ccc|c} | & \cdots & | & \\ u_1 & \cdots & u_d & \cdots \\ | & \cdots & | & \end{array}\right) \text{ and } V := \left(\begin{array}{ccc|c} | & \cdots & | & \\ v_1 & \cdots & w_d & \cdots \\ | & \cdots & | & \end{array}\right).$$

Then with $R = V \left(\begin{array}{c|c} \mathbb{I}_d & 0 \\ \hline 0 & 0 \end{array}\right) U^T, \text{span}(Z) \subseteq \text{span}(RA)$.

Corollary 6 (Representational power of GAEs). For any (trained) graph auto-encoder in $\mathcal{Z}_{\text{relu}} \supseteq \mathcal{Z}_{\text{relu},X}$, there exists an equivalent, featureless linear encoder in $\mathcal{Z}_{\text{lin}}$, that can achieve the same training loss: $\mathcal{Z}_{\text{lin}} \supseteq \mathcal{Z}_{\text{relu}}$.

Proof of Corollary 6. Let $Z_{\text{lin}}(W) = \tilde{A} \cdot W$ for $W \in \mathbb{R}^{n\times f}$ be the latent representation of the linear model with weights W. We can write the latent representation for any GAE as a linear function of the form $Z_{\text{relu}}(W) = \tilde{A} \cdot W$ for some matrix $W \in \mathcal{W}$, where $\mathcal{W}$ is the set of matrices that can be represented by the relu term in the function: $\mathcal{W} = \{\text{relu}(\tilde{A}XW^{(0)})W^{(1)} : W^{(0)} \in \mathbb{R}^{n\times d}, W^{(1)} \in \mathbb{R}^{d\times f}\} \subseteq \mathbb{R}^{n\times f}$. We now define the possible latent embeddings that the two models can learn. For the relu encoder we have $\mathcal{Z}_{\text{relu}} = \{Z_{\text{relu}}(W) \text{ for all } W \in \mathcal{W}\}$, and for the linear model we get $\mathcal{Z}_{\text{lin}} = \{Z_{\text{lin}}(W) \text{ for all } W \in \mathbb{R}^{n\times g}\}$. Since $\mathcal{W} \subseteq \mathbb{R}^{n\times g}$, the set of learnable embeddings of the relu encoder is a true subspace of all learnable embeddings of the linear encoder: $\mathcal{Z}_{\text{relu}} = \{\tilde{A}W : W \in \mathcal{W}\} \subseteq \{\tilde{A}W : W \in \mathbb{R}^{n\times g}\} = \mathcal{Z}_{\text{lin}}$. It follows that, given any loss l, the linear model can achieve at least as good performance as the relu encoder: $\inf_{Z_{\text{relu}}\in\mathcal{Z}_{\text{relu}}} l(Z_{\text{relu}}) \geqslant \inf_{Z_{\text{lin}}\in\mathcal{Z}_{\text{lin}}} l(Z_{\text{lin}})$.

Consequences. To better understand the implications of Corollary 6, it is instructive to engage in the following thought experiment. Consider the standard supervised learning setting with a training data matrix $U \in \mathbb{R}^{n\times d}$ and labels $y \in R$. For any function (say a deep neural network) $f : \mathbb{R}^{n\times d} \to \mathbb{R}^n$, it is possible to express $f(U) = UU^+f(U) = UW$, for some $W \in \mathbb{R}^{d\times 1}$ if and only if U has a right inverse. This is a mild requirement if $d > n$. Note that this is precisely the "underdetermined" or "high-dimensional" setting where it is possible to find a linear map that perfectly fits the training data to the function values. If $d < n$, the right inverse cannot exist; therefore, one cannot find such a W. Let us contrast this with our setting of GNNs in Corollary 6. Since we consider graph networks, the input is the adjacency matrix $\tilde{A} \in \mathbb{R}^{n\times n}$; that is, we have feature dimensions equal to sample size ($n = d$). If $\tilde{A}$ is full rank, one can similarly find a linear map that perfectly fits the outputs of any nonlinear mapping of the input data. While it may seem surprising at first glance, it is intuitively clear that in high dimensions, one can always find a linear map that perfectly fits the outputs of any nonlinear mapping. If the model is restricted to the input data domain, we can attain the same "training loss". But even if $\tilde{A}$ was not full rank, in the graph auto-encoder setting, the GCN's last layer has linear activation, restricting the solution space in exactly the same way the linear model does.

B Experimental details

For our empirical evaluation we consider one synthetic dataset and three real world citation networks; `Cora`, `Citeseer` and `Pubmed`. Table 1 shows the graph statistics for the considered datasets.

dataset	nodes	edges	density	dim. features
`Cora`	2708	5278	0.00143	1433
`Citeseer`	3327	4614	0.00083	3703
`Pubmed`	19717	44324	0.00023	500
Synthetic	~ 1000	$\sim 10000 - 20000$	$\sim 0.01 - 0.02$	64

Table 1: Graph statistics of the real world datasets and averaged values in the synthetic setting.

For every dataset we compute the performances for two different tasks; node and link prediction, and four different models: using the relu encoder and the linear encoder once with features and also without features. So we compute performances in eight different settings for each dataset.

Every graph is split into train, validation and test sets with a ration of 70/10/20. So training always happens on either 70% of the edges and all nodes for the link prediction task, or on only a set of 70% of the nodes for the node prediction task. Test performance is then evaluated on 90% of the edges / nodes. We do not use the parts of the graph used for validation.

The code for the relu graph auto-encoder is publicly available with an MIT license.

We run the experiments on an internal cluster on *Intel XEON CPU E5-2650 v4* and *GeForce GTX 1080 Ti*. All experiments on the synthetic dataset take about 9 hours on single CPU and single GPU. Experiments for `Cora` and `Citeseer` take about 4 and 5 hours respectively. For the `Pubmed` dataset, which is the largest one, running all 8 setups took about 4 days on a single CPU and two GPUs.

C Additional experiments

For the real world datasets and the link prediction task we show result for embedding into 4 dimensions in the main paper. Table 2 show results when embedding into 16 dimensions. All models show near to optimal performance which indicates that embedding into 16 dimensions is barely a restriction and too simple of a task.

	cora			
	edge		node	
	linear	GAE	linear	GAE
features	0.91 ± 0.0088	0.88 ± 0.0123	0.99 ± 0.0009	0.99 ± 0.0014
featureless	0.84 ± 0.0078	0.85 ± 0.0122	0.98 ± 0.0027	0.98 ± 0.0016
	citeseer			
	edge		node	
	linear	GAE	linear	GAE
features	0.92 ± 0.0087	0.84 ± 0.0264	0.99 ± 0.0010	0.99 ± 0.0012
featureless	0.78 ± 0.0131	0.78 ± 0.0149	0.98 ± 0.0015	0.98 ± 0.0014
	pubmed			
	edge		node	
	linear	GAE	linear	GAE
features	0.96 ± 0.0018	0.92 ± 0.0037	0.98 ± 0.0006	0.97 ± 0.0055
featureless	0.83 ± 0.0056	0.83 ± 0.0049	0.99 ± 0.0009	0.99 ± 0.0006

Table 2: Test performance in terms of AUC for the three real world datasets `Cora` `Citeseer` and `Pubmed` with given standard deviations. We consider relu and linear encoder models with and without features. Top row: Link prediction task, points are embedded into 16 dimensions. Bottom row: Node prediction task, points are embedded into 16 dimensions.

Chapter 3

Discussion

Unsupervised machine learning on graphs represents a frontier in discovering hidden patterns and relationships within complex data. Unlike traditional supervised methods that rely on labeled data, unsupervised learning allows us to explore and uncover the intrinsic structure of graphs without predefined outcomes. This capability is particularly powerful in real-world scenarios, such as social networks, biological systems, and recommendation engines, where labeled data is often scarce or expensive to obtain. By harnessing the power of unsupervised techniques, we can reveal community structures, detect anomalies, and understand the underlying geometry of data, opening new avenues for innovation and deeper insights into the interconnected world around us.

In this thesis, we present two distinct perspectives on the field of unsupervised machine learning on graphs. Through these lenses, we demonstrate the versatility and power of unsupervised methods, as well as their challenges.

The first perspective explores tangles, an object from mathematical graph theory, and its suitability for clustering. With this contribution, we showcase an unsupervised technique that inherently allows for interpretable outcomes that are additionally soft and hierarchical. While the algorithmic framework only requires a set of cuts of the dataset and some notion of similarity between objects, it can include domain knowledge in different steps of the process so that experts can tune the algorithm specifically to each problem and any data.

The second perspective delves into linear and non-linear graph auto-encoders, further expanding our understanding of their biases in graph representation learning. Our evaluations support the idea that the bias introduced by the non-linearity is not crucial to meaningfully reducing the solution space. On the other hand, features can introduce a powerful inductive bias to both encoder architectures, improving their test performance.

While these findings mostly advance the theoretical understanding of graph-based learning, our tangles approach has practical implications for clustering not only in domains

where graph structures are prevalent but in any data with some similarity measure.

In both cases, we highlight the limitations of the methods, which provide opportunities for further exploration. For the tangles framework, we are sure that our current approach does not fully leverage all the information in the tangle search tree. We believe there are different, maybe better, ways to extract the clustering information, which could, in return, put less weight on the choice of the hyperparameters and make the approach even more powerful for explorative research, potentially mitigating some of the challenges we encountered, as the gap problem. For linear graph auto-encoders, we see the need to explore this phenomenon further on different types of graphs and different (downstream) tasks. This might give us a deeper understanding of the information used. For the linear model, it might also be fruitful to investigate the explainability of the approach, as linear models often allow us to determine the influence of the input data directly. However, for graphs, it is not always intuitive what the influence of a certain combination of nodes in the input means. Further exploration might shed some light on this question.

In summary, we underscore the significance of unsupervised learning on graphs as a powerful tool for understanding complex data relationships. We question whether complex non-linear models like graph neural networks are always necessary, giving a set of functions where linear models can be just as powerful. In contrast, with the tangle framework, we provide a systematic and interpretable approach that opens new pathways for research and application in science.

Bibliography

S. J. Ahn and M. Kim. Variational graph normalized autoencoders. Association for Computing Machinery (ACM), 2021.

V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

K. E. Burger, S. Klepper, U. von Luxburg, and F. Baumdicker. Inferring ancestry with the hierarchical soft clustering approach tanglegen. *Genome Research*, 2024.

T. R. Davidson, L. Falorsi, N. De Cao, T. Kipf, and J. M. Tomczak. Hyperspherical variational auto-encoders. Uncertainty in Artificial Intelligence (UAI), 2018.

K. Do, T. Tran, and S. Venkatesh. Graph transformation policy network for chemical reaction prediction. In *Conference on knowledge discovery and data mining (KDD)*, 2019.

W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973.

S. V. Dongen. Graph clustering by flow simulation, 2000.

L. Euler. Solutio problematis ad geometriam situs pertinensis. *Commentarii Academiae Scientarum Imperialis Petropolitanae*, pages 128–140, 1736.

M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23 (2):298–305, 1973.

Z. Gao, C. Jiang, J. Zhang, X. Jiang, L. Li, P. Zhao, H. Yang, Y. Huang, and J. Li. Hierarchical graph learning for protein–protein interaction. *Nature*, 14(1):1093, 2023.

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. International Conference on Machine Learning (ICML), 2017.

A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. International Conference on Knowledge Discovery and Data Mining (KDD), 2016.

S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. Conference on artificial intelligence (AAAI), 2019.

W. L. Hamilton. *Graph representation learning*. Morgan & Claypool Publishers, 2020.

K. Jha, S. Saha, and H. Singh. Prediction of protein–protein interaction using graph neural networks. *Nature*, 12(1):8360, 2022.

R. Jiang, Z. Wang, J. Yong, P. Jeph, Q. Chen, Y. Kobayashi, X. Song, S. Fukushima, and T. Suzumura. Spatio-temporal meta-graph learning for traffic forecasting. Conference on artificial intelligence (AAAI), 2023.

H. Kim, B. S. Lee, W.-Y. Shin, and S. Lim. Graph anomaly detection with graph neural networks: Current status and challenges. *IEEE Access*, 10:111820–111829, 2022.

T. N. Kipf and M. Welling. Variational graph auto-encoders. *NeurIPS Workshop on Bayesian Deep Learning*, 2016.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. International Conference on Learning Representations (ICLR), 2017.

S. Klepper. Towards understanding diffusion models (on graphs). *arXiv:2409.00374*, 2024.

S. Klepper and U. von Luxburg. Relating graph auto-encoders to linear models. *Transactions on Machine Learning Research (TMLR)*, 2023.

S. Klepper, C. Elbracht, D. Fioravanti, J. Kneip, L. Rendsburg, M. Teegen, and U. Von Luxburg. Clustering with tangles: Algorithmic framework and theoretical guarantees. *Journal of Machine Learning Research (JMLR)*, 24(190):1–56, 2023.

X. Li, H. Zhang, and R. Zhang. Adaptive graph auto-encoder for general data clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12): 9725–9732, 2022.

Y. Li, R. Shafipour, G. Mateos, and Z. Zhang. Supervised graph representation learning for modeling the relationship between structural and functional brain connectivity. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2020.

Y. Long, M. Wu, Y. Liu, Y. Fang, C. K. Kwoh, J. Chen, J. Luo, and X. Li. Pre-training graph neural networks for link prediction in biomedical networks. *Bioinformatics - Oxford University Press*, 38(8):2254–2262, 2022.

M. E. Newman. Modularity and community structure in networks. National academy of sciences, 2006.

M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

S. Peng, Y. Zhou, L. Cao, S. Yu, J. Niu, and W. Jia. Influence analysis in social networks: A survey. *Journal of Network and Computer Applications*, 106:17–32, 2018.

B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. International conference on Knowledge discovery and data mining (KDD), 2014.

G. Peters, F. Crespo, P. Lingras, and R. Weber. Soft clustering–fuzzy and rough approaches and their extensions and derivatives. *International Journal of Approximate Reasoning*, 54(2):307–322, 2013.

M. R. Quinn and T. M. McQueen. Identifying new classes of high temperature superconductors with convolutional neural networks. *Frontiers in Electronic Materials*, 2: 893797, 2022.

P. Reiser, M. Neubert, A. Eberhard, L. Torresi, C. Zhou, C. Shao, H. Metni, C. van Hoesel, H. Schopmans, T. Sommer, et al. Graph neural networks for materials science and chemistry. *Communications Materials*, 3(1):93, 2022.

S. Rhee, S. Seo, and S. Kim. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. International Joint Conference on Artificial Intelligence (IJCAI), 2018.

T. Sarkar, A. Sharma, A. K. Das, D. Deodhare, and M. D. Bharadwaj. A neural network based approach to predict high voltage li-ion battery cathode materials. IEEE Conference on Devices, Circuits and Systems, 2014.

K. Sun, L. Wang, B. Xu, W. Zhao, S. W. Teng, and F. Xia. Network representation learning: From traditional feature learning to deep learning. *IEEE Access*, 8:205600–205617, 2020.

R. Van Den Berg, N. K. Thomas, and M. Welling. Graph convolutional matrix completion. International Conference on Knowledge Discovery and Data Mining (KDD), 2017.

P. Veličković. Theoretical foundations of graph neural networks. https://petar-v.com/talks/GNN-Wednesday.pdf. Accessed: 2024-09-05.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. International Conference on Learning Representations (ICLR), 2018.

U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17:395–416, 2007.

Y. Wang, J. Zhang, S. Guo, H. Yin, C. Li, and H. Chen. Decoupling representation learning and classification for gnn-based anomaly detection. pages 1239–1248. Special interest group of information retrieval (ACM SIGIR), 2021.

Z. Weng, W. Zhang, and W. Dou. Adversarial attention-based variational graph autoencoder. *IEEE Access*, 8:152637–152645, 2020.

B. Wu, S. Han, K. G. Shin, and W. Lu. Application of artificial neural networks in design of lithium-ion batteries. *Journal of Power Sources*, 395:128–136, 2018.

F. Wu, B. A. Huberman, L. A. Adamic, and J. R. Tyler. Information flow in social groups. *Physica A: Statistical Mechanics and its Applications*, 337(1-2):327–335, 2004.

Q. Wu, H. Zhang, X. Gao, P. He, P. Weng, H. Gao, and G. Chen. Dual graph attention networks for deep latent representation of multifaceted social effects in recommender systems. The world wide web conference (WWW), 2019.

Y. Wu, H.-N. Dai, and H. Tang. Graph neural networks for anomaly detection in industrial internet of things. *IEEE Internet of Things Journal*, 9(12):9214–9231, 2021.

S. Xiao, S. Wang, and W. Guo. Sgae: Stacked graph autoencoder for deep clustering. *IEEE Transactions on Big Data*, 9(1):254–266, 2023.

L. Xie, D. Pi, X. Zhang, J. Chen, Y. Luo, and W. Yu. Graph neural network approach for anomaly detection. *Measurement*, 180:109546, 2021.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? International Conference on Learning Representations (ICLR), 2018.

S. Yu, F. Xia, J. Xu, Z. Chen, and I. Lee. Offer: A motif dimensional framework for network representation learning. International Conference on Information and Knowledge Management, 2020.

D. Zhang, J. Yin, X. Zhu, and C. Zhang. Network representation learning: A survey. 2018.

X.-M. Zhang, L. Liang, L. Liu, and M.-J. Tang. Graph neural networks and their current applications in bioinformatics. *Frontiers in genetics*, 12:690049, 2021.

M. Zitnik, M. Agrawal, and J. Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018.