Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

# Master Thesis Computer Science

## Tangles on Ordinal Data

Alexander Conzelmann

Datum

**Reviewers**

Prof. Dr. Ulrike von Luxburg
Theory of Machine Learning
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Prof. Felix Wichmann, DPhil
Neural Information Processing
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

**Conzelmann, Alexander**
*Tangles on Ordinal Data*
Master Thesis Computer Science
Eberhard Karls Universität Tübingen
Thesis period: 04/2022-10/2022

# Abstract

We investigate a new algorithm for clustering triplet data (comparison-based data without absolute distance information). This new approach is based on the tangles framework, a tool previously used to finde dense structures in graphs, which we extend to work on triplet data. Current work focusses on embedding triplet data into a euclidean space, resulting possibly in distortions of our data, and then following up with a classical clustering algorithm. In contrast, our proposed method does not construct an intermediate embedding, potentially introducing less distortions on our data and achieving a higher clustering accuracy. Additionally to being competitive in performance, our approach provides both explainability as well as a cluster hierarchy on top with no added cost. We validate the tangles algorithm both on synthetic data under a diverse range of external influences as well as experimental data from the realm of psychophysics.

# Zusammenfassung

Bei einer englischen Masterarbeit muss zusätzlich eine deutsche Zusammenfassung verfasst werden.

# Acknowledgements

Write here your acknowledgements.

# Contents

# Chapter 1

# Introduction

We consider the task of clustering data $X = \{x_1, x_2, \dots x_n\}$ for which neither explicit features nor concrete distances between the data points are known to us. The only form of information we have on the data are so-called *triplet comparisons* or *triplets* for short. A triplet on $X$ is written as $(x_i, x_j, x_k)$ and tells us that $x_i$ is closer to $x_j$ than to $x_k$. This problem setting often arises when one works with data from human observers. An example, investigated among others by (Shepard, 1962), is human color perception. It would be hard for human observers to accurately rate colors in terms of features, let alone define sensible features in the first place. It would also be hard to rate colors in terms of concrete distances to each other. Additionally, the experiment designer would have to deal with uniting differing internal scales of different observers during data evaluation. A preferred approach might be to gather triplets on the data. To obtain these triplets, the experiment designer can repeatedly draw three different colors, which are presented to a human observer, together with a suitable question. For example, we might draw the colors violet, red, yellow, and ask the the observer *is violet more similar to red or to yellow?* These questions are comparatively easy to answer for human observers. What remains is the question of how to evaluate the obtained triplet data.

A small research community has formed around the task of dealing with triplets. Most of this community focusses on ordinal embeddings (Agarwal et al., 2007, Tamuz et al., 2011, L. van der Maaten and K. Weinberger, 2012, Terada and Luxburg, 2014, Jain et al., 2016, Ghosh et al., 2019, Anderton and Aslam, 2019). An ordinal embedding is an algorithm that aims to place the data points into a euclidean space such that the euclidean distances between the embedded points respects as many of the original triplet comparisons as possible. However, this approach is not always perfect: we often cannot satisfy all triplet comparisons, no matter the dimension of the embedding space or how the points are placed in it. This is because the euclidean distance is a proper metric, so it is symmetrical and has to obey the triangle inequality, limiting its flexibility.

A big advantage of ordinal embeddings however is precisely that we obtain a euclidean representation of the data. For euclidean data, there are many good, readily available algorithms for almost all tasks. Thus, clustering on triplet data can be tackled by getting a euclidean representation of the data from an ordinal embedding and applying a classical clustering algorithm (such as k-Means) on this representation. This approach was for example demonstrated as a baseline in (Kleindessner and von Luxburg, 2017). However, as mentioned before, an ordinal embedding often cannot satisfy all triplet comparisons on the original data, and is therefore not necessarily a faithful representation. An alternative approach is devising an algorithm to solve the desired task directly using the triplets, which has shown promising results. Kleindessner and von Luxburg (2017) estimated the lens-depth function of the data using triplets (of a slightly altered format) and used this for medoid estimation, outlier identification, clustering and classification. Kleindessner and von Luxburg (2017) constructed a kernel function from the triplets, which they demonstrated to work well with different kernel-based clustering algorithms. Ghoshdastidar et al. (2019) used the triplets to estimate a similarity function between the data points and applied a linkage algorithm to obtain a hierarchical clustering from the data.

In a recent work, Klepper et al. (2020) proposed a novel framework for clustering based on tangles, which are a mathematical tool originating from graph theory (Robertson and Seymour, 1991). The tangles algorithm has been shown to have interesting properties, such as inherent explainability (under certain conditions) and being suitable for hierarchical clustering. One of the central objects in this framework are cuts, ways of dividing a set into two distinct, non-overlapping subsets. To cluster data using the tangles framework, we first need to obtain a set of cuts on the data. We additionally require that these cuts hold a little bit of information about the cluster structure of the data. The tangles framework can then aggregate the information contained in these cuts to a clustering.

In this thesis, we present two novel methods to generate cuts suitable for the tangles algorithm out of triplets. Using these methods, we demonstrate that the tangles algorithm can be successfully applied to (hierarchically) cluster triplet data without creating an intermediate ordinal embedding. We evaluate our approach by simulated and experimental data and show that it is competitive in performance to ordinal embedding based approaches, while providing explainable results.

This thesis is organized as follows: In Chapter 2, we give an introduction to tangles and ordinal data, which lays the necessary foundations for the rest of the work. In Chapter 3, we present our two cut finding algorithms, named *landmark cuts* and *majority cuts*. We use simulated data in Chapter 4 to show the performance of tangles using our cuts under different circumstances, such

as the noise level or availability of the triplets. In Chapter 5 we showcase an example evaluation with real data from the domain of psychophysics to establish tangles as a practical tool for triplet data. While doing so, we also highlight the inherent explainability of the clustering produced by the tangles algorithm.

# Chapter 2

# Theoretical Background

In this chapter we will a brief introduction to the theoretical concepts used in this work. In particular, we will give an in-depth explanation about tangles, triplet data, and state-of-the-art methods of evaluating triplet data.

## 2.1 Tangles

Tangles have been used in mathematical graph theory to study highly cohesive structures, introduced originally by Robertson and Seymour (1991). Recently, interesting areas of application have been proposed: Diestel and Whittle (2017) makes a proof of concept that tangles could be used in image analysis. Diestel (2019) describes how tangles can be used in social sciences, for example to identify different mindsets in people answering questionnaires. Fluck (2019) shows that, under specific circumstances, tangles can be used to reconstruct the dendrogram in a hierarchical clustering setting.

In recent times, Klepper et al. (2020) describes a very flexible setup, where tangles are successfully applied to solve problems of clustering. The mentioned work develops an algorithmic framework and gives theoretical guarantees for basic problem settings. Additionally, it introduces simplified notations, adapted to the domain of computer science. When talking about tangles, we will use the definitions introduced therein.

In this section, we now give an introduction to the basic notions, theory and applications of tangles in a clustering context. For in-depth explanations of the algorithms and exact procedures, refer to Klepper et al. (2020).

### 2.1.1 Definitions

**Cuts.** The central object in tangles theory is a *cut* (also referred to as a bipartition in other works). A cut is a division of a set $V = \{v_1, v_2, \ldots\}$ into

two distinct subsets $A, B \subset V$, such that

$$A \cap B = \emptyset \text{ and } A \cup B = V.$$

We can also write a cut as $P = \{A, \overline{A}\}$, with $A \subset V$ and $\overline{A}$ being the complement of $A$ with respect to $V$.

**Cost function.** For a cut to be useful in clustering, we expect it to hold some degree of information about the cluster structure of our data. This means that an informative cut should not separate groups of data that are tightly coupled. On a graph, an informative cut $P = \{A, \overline{A}\}$ might separate the set of nodes $V$ such that there are only a few edges between $A$ and $\overline{A}$. How informative a cut is, is quantified by a *cost function* $c : \mathcal{P}(V) \to \mathbb{R}$, with $\mathcal{P}(V)$ denoting the power set of V. One needs to choose an appropriate cost function beforehand and the performance of tangles will also depend on how well the cost function and the problem setting fit together. For example, on an unweighted graph we might want to choose $c(P)$ as the number of edges going between the nodes of the two sets $A, \overline{A}$.

**Orientations.** An *orientation* of a cut means that we pick a specific side of a cut. An *orientation* of a set of cuts $\mathcal{B} = \{\{A_1, \overline{A_1}\}, \ldots, \{A_n, \overline{A_n}\}\}$ is then a set of orientations of cuts $O = \{o_1, o_2, \ldots o_n\}$ where $o_i$ corresponds to either the partition $A_i$ (oriented *left*) or $\overline{A_i}$ (oriented *right*).

**Tangles and agreement.** Assume that for a set of elements $V$ we have a set of cuts $\mathcal{B} = \{\{A_1, \overline{A_1}\}, \ldots, \{A_n, \overline{A_n}\}\}$ on $V$. This set of cuts, together with the cost function, should tell us a lot about the cluster structure of the data: for all cuts, we know how much they do or don't separate dense regions in $V$. This information in the cuts is aggregated and brought into a useable form by the tangles framework. For this, we find in $\mathcal{B}$ the set of consistent orientations (also called *tangles*). These correspond to specific ways of orienting the cuts such that they point to cohesive structures in the data. A tangle is an an orientation for which:

$$\forall A, B, C \in O : |A \cap B \cap C| \geq a \tag{1}$$

for some fixed parameter $a \in \mathbb{N}$, which we refer to as *agreement* parameter. Equation 1 is also called the *consistency condition*.

**Order.** With our definition of tangles, a lot of sets of cuts wouldn't allow for any tangles, as there are too many cuts to consistently orient them. Imagine that a set of cuts would contain a few random cuts. In expectation, each of these halve our set of points, so we can at most orient on the order of $O(\log(n))$

many random cuts consistently. This is resolved using the cost function: one restricts the tangles to a set of low-cost (and thus very insightful) cuts $P_\psi$, using a threshhold $\psi \in \mathbb{R}$ such that

$$P_\psi = \{c(P) \leq \psi\}.$$

A tangle on $P_\psi$ is said to have *order* $\psi$.

To illustrate some of the concepts better, we have included a schematic drawing of a tangle on a simple data set composed of two clusters in Figure 2.1. Here, we might already gain some intuition on why tangles are able to find dense structures in data. The tangle that is depicted in the figure orients all cuts to the left side (indicated by the arrows), so that they point towards the cluster on the left. Another possible tangle might orient all cuts to the right, pointing to the right cluster. Notice that a tangle on this set of cuts can only either orient all cuts to the left or to the right, else the consistency criterion is violated. All in all, there is exactly one tangle for each cluster.

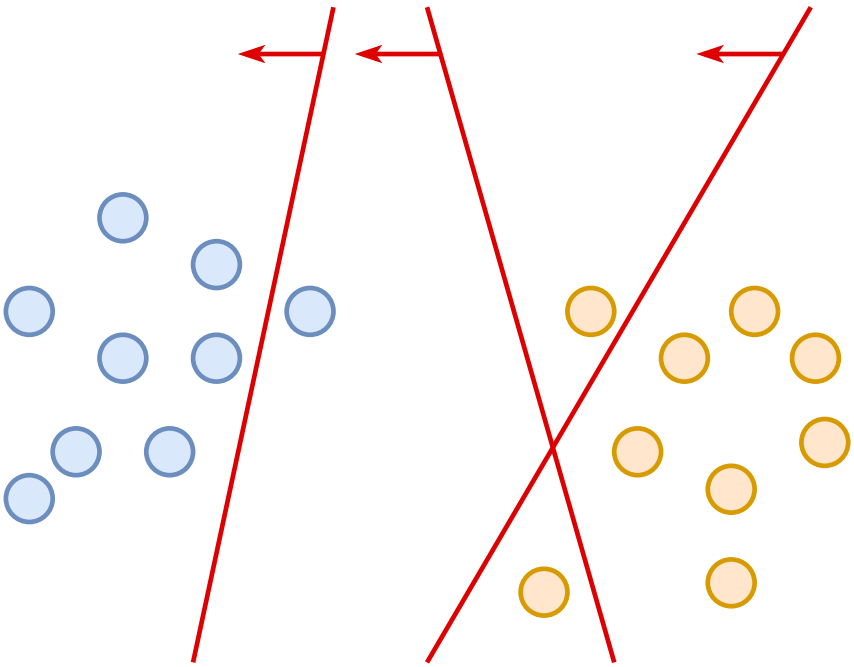


**Figure 2.1:** An example of how a simple tangle might look like if we assume a reasonably sized agreement (say $a = 3$). The data set consists of two clusters, one left (blue), one right (orange). We assume that we have obtained three cuts on the data set, represented by red lines. We draw a possible orientation on the cuts, indicated by the red arrows on them. The entirety of all orientations makes up one possible tangle for the cuts on this data set.

### 2.1.2 Processing tangles to a clustering

In Figure 2.1, each of the tangles we found clearly pointed in the direction of exactly one cluster. However, tangles on real data sets are usually much more complex. In this section, we explain an algorithmic procedure how clusters can be identified through tangles. We assume that we first have chosen an appropriate cost function $c$ and an agreement parameter $a$.

We assume that we are given a set of cuts $\mathcal{B} = \{\{A_1, \overline{A_1}\}, \ldots, \{A_n, \overline{A_n}\}\}$ which are sorted in ascending order according to the cost function $c$. Next, we build a tree structure on the set of these cuts, the so-called *tangle search tree*. In the tree, the value of each node of level $i$ is an orientation on the set of cuts $\mathcal{B}_{1:i} = \{\{A_1, \overline{A_1}\}, \ldots \{A_i, \overline{A_i}\}\}$, with the value of $n_0 = \emptyset$. We build the tree in an iterative manner: to determine the nodes of level $k$, we iterate through all nodes $n_j$ of level $k-1$. We then try to add the cut $\{A_k, \overline{A_k}\}$ in left orientation to $n_j$. If this orientation is consistent with respect to $a$, we add the orientation $n_j \cup A_k$ as a left child. We then try to add the cut in right orientation as well, and append $n_j \cup \overline{A_k}$ as right child if it is consistent. By construction, each node in the tangle search tree then represents a tangle, and every level of the tree contains all possible tangles of threshhold $\psi_k$ which directly corresponds to the cost of the $k$-th cut $c(\{A_k, \overline{A_k}\})$.

An exemplary tangle search tree is illustrated in Figure 2.2. By the construction above, we can now determine the value of each node. As an example for the node in level 3, we start with $\emptyset$ at the root node. To get to the node, we have to go right from the root, adding cut $P_1$ in a right-oriented way. We then go right again for $P_2$, and left for $P_3$. Thus, we know that the node (and the corresponding tangle) in level 3 has the value $T = \{\overline{A_1}, \overline{A_2}, A_3\}$. By the construction of the tangle tree, we also know that this tangle is now the only one on on the set $\{P_1, P_2, P_3\}$.

We will now try to understand how to use the tangle search tree for clustering. First, observe that a cheaper cut cuts through more loosely connected structures of the data set, while a more expensive cut can cut through more densely connected structures. Thus, in the tangles search tree we will start with coarse divisions of our data at the root, and proceed to finer divisions as we go down the tree. With respect to clustering, the interesting nodes in the tangle tree are the *leaves* and the *splitting nodes* (nodes with two children).
**Leaves** are the final clusters, as we cannot add more cuts to the tangles and thus cannot subdivide the structure the tangle points to further. In the end, we will have exactly one cluster for each leaf node.
**Splitting nodes** represent meaningful paths in our tree. If a node only has one children, the cut that was added last only further restricts the data set. If a node has two children, the cut however presents a decision, as we can either follow the left or the right child of the cut when deciding how to subdivide our cluster further.

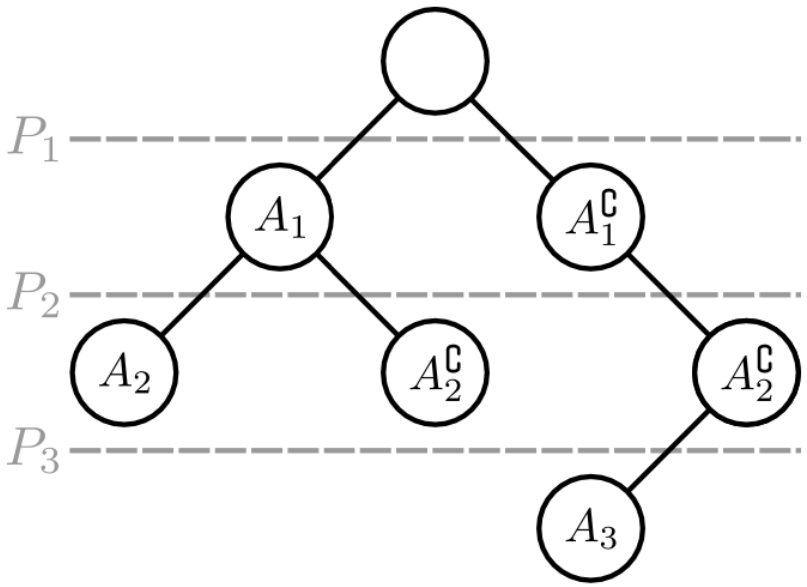



**Figure 2.2:** An example of a possible tangles search tree for a set of cuts $\mathcal{B} = \{\{A_1, \overline{A_1}\}, \{A_2, \overline{A_2}\}, \{A_3, \overline{A_3}\}\}$. Each level corresponds to the tangles of the order given by the cut $P_i$ that is written on the dashed line to the left of it. Figure taken with permission from Klepper et al. (2020).

For the splitting nodes, we are interested in which cuts are really the ones determining which cluster a point belongs to. Naively, we could use only the cut that produced the split, and assign all points that are contained in the left orientation of the cut to the left child and all that are contained in the right orientation to the right child. This however seems to waste information contained in the cuts further down the tree. We thus determine determine the set of *characterizing cuts*. A cut belongs to this set, if it is both oriented the same way inside the subtrees, and oriented in a different way between the two subtrees. Thus, all characterizing cuts make a meaningful decision between left and right subtree and are coherent in their decision inside the subtree. To illustrate this, we take a look at the exemplary tree in Figure 2.2. Here, for the root node, $P_1$ is a characterizing cut, while $P_2$ is not: below the node $A_1$, the cut is both oriented to the left and to to the right, violating the requirement that the cuts are always oriented the same way inside the subtrees.

With this knowledge, we can now obtain a soft, hierarchical clustering from the tangles search tree. *Soft* means that every data point is assigned a probability of belonging to each cluster. *Hierarchical* means that we have a hierarchy on the resulting clusters, which arises naturally from the form of the tangles search tree. For a soft clustering, we determine with what probability a point $x$ belongs to a cluster $C$. To do this, we start from the root. At every splitting node of the tree (which includes the root), we examine the

set of characterising cuts. We then orient them in the same way they are oriented in the left subtree and count how many of these so oriented cuts contain $x$. Divided by the total amount of characterising cuts at the splitting node, we receive the probability $p_L$ that $x$ belongs to the left cluster. As the characterising cuts are always oriented differently in the two subtrees, the probability of $a$ belonging to the right cluster is then given by $1 - p_L$. We can include these probabilites on the edges of our tree. To find out with what probability $x$ belongs to $C$, we now take the product of all edge probabilities on the path from the root to the leaf node that corresponds to $C$. By assigning each node to the cluster it belongs to with the highest probability, we can also obtain the corresponding *hard* clustering.

## 2.2 Ordinal constraints and triplet data

Assume that we have a set of objects for which we don't know absolute distance information between them. A dataset of ordinal constraints is then a set of comparisons on these objects such as *item $i$ is closer to item $j$ than to item $k$*. Formally, we assume that $i, j, k$ are from a set $D$ where we can define a dissimilarity function $d : D \times D \to \mathbb{R}$. Note that $d$ can be a proper metric on $D$, but does not have to be. Using $d$, we can express the constraint *item $i$ is closer to item $j$ than to item $k$* as $d(i, j) < d(i, k)$. Such data is often encountered when humans are asked to judge objects, as they naturally are better at comparing objects than at accurately placing them on an abstract scale (Demiralp et al., 2014). Applications consist of estimating perceptual scales in psychophysics (Haghiri et al., 2020) or crowd-sourcing clustering algorithms (Ukkonen, 2017). We focus mainly on the realm of psychophysics.

Ordinal constraints are usually presented in one of two forms: quadruplets (used for example in Ghoshdastidar et al. (2019)) and triplets (used for example in Vankadara et al. (2021), Haghiri et al. (2019)). Let $D$ be a set of objects, and $a, b, c, d \in D$, and $d$ be a dissimilarity function on $D$. A quadruplet (a,b,c,d) expresses the following constraint on our data points:

$$\mathrm{d}(a, b) < \mathrm{d}(c, d).$$

Analogously, a triplet (a,b,c) expresses

$$\mathrm{d}(a, b) < \mathrm{d}(a, c).$$

A triplet (a,b,c) can also be expressed by the quadruplet (a,b,a,c), making quadruplets strictly more general.

Datasets of triplets (which we also simply call triplet data) are almost always obtained by asking humans participants. For example, we might collect triplet data on images by presenting human participants with three images

$a, b, c$, ask them: *is a more similar to b or c?*. The way that this question is formulated varies on the context of the experiment (and might also influence their answers). Other possible experiment setups are for example presenting the participant with three images, and asking *which is the most central image?*, or *which is the odd one out?*, but the results can then always transformed back to triplet format for further prcoessing. For example, if $a$ is the *odd-one-out* of the three elements $a, b, c$, then we know that $d(b, c) < d(b, a)$ and $d(c, b) < d(c, a)$.

## 2.3 Algorithms on triplet data

Most of the algorithms that the machine learning community uses require euclidean data ($k$-Means, support vector machines, neural networks, et cetera...). Triplet data is an unusual format, which is why one of the most common evaluation methods is to first use an ordinal embedding algorithm on the triplet data to transform it into euclidean space, before processing it further. We therefore divide the algorithmic approaches presented in this section into two parts, ordinal embeddings, and other algorithms, which achieve end tasks directly without applying an ordinal embedding beforehand. The latter category is where the tangles algorithm belongs to. Not using an ordinal embedding as a first processing step can have distinct advantages: we introduce additional distortions to the data and we avoid eventual biases that the ordinal embedding algorithms might have.

### 2.3.1 Ordinal embeddings

One of the most central problems when dealing with triplet data consists of finding a so called *ordinal embedding* of the data. If we have a set of triplet comparisons $T = \{t_1, t_2, \dots t_n\}$, of the form $t_i = (a, b, c)$, encoding that $d(a, b) < d(a, c)$, we want to find a set of points $y_1, y_2, \dots y_n \in \mathbb{R}^m$, such that they uphold most of the original triplet constraints in $\mathbb{R}^m$ with the euclidean distance as metric. Formally, we want to minimize (Vankadara et al., 2021)

$$\min_{y_1,\dots y_n \in \mathbb{R}^m} \sum_{t=(i,j,k)\in T} \mathbb{1}_{\|y_i - y_j\|_2 < \|y_i - y_k\|_2}.$$

This problem is difficult to optimize and thus various algorithms have been proposed that solve a relaxed or modified version of this objective function. The algorithms are mostly formulated for quadruplets, as this is more general: we can convert triplets to quadruplets but not necessarily vice versa.

- Soft Ordinal Embedding (SOE, Terada and Luxburg, 2014) introduces a scale parameter $\delta$ and not only punishes violated ordinal constraints

with a binary value, but also by how much they are violated using the actual distance between embedded points. The authors assume we have a set of quadruplets $Q$ on our data set. They propose the following error function, which their algorithm minimizes:

$$\mathrm{Err}_{\mathrm{soft}}(X \mid \delta) = \sum_{i<j} \sum_{k<l} o_{i,j,k,l} \max[0, d_{ij}(X) + \delta - d_{kl}(X)],$$

where $X$ is the embedding of the points, $d_{ij}(X)$ is the euclidean distance of points with index $i$ and $j$ in $X$, $\delta$ is a scale parameter and $o_{i,j,k,l}$ is 1 if $i$ is closer to $j$ than $k$ to $l$ according to $Q$, and 0 else.

- Generalized Non-Metric Multidimensional Scaling (GNDMS, Agarwal et al., 2007) finds a gram matrix of an embedding. They cast the constraints given by a set of quadruplets $Q$ as constraints on the gram matrix and use these as inequalities in a constrained optimization problem. The dimension of the embedding is controlled via a regularization parameter $\lambda$ on the trace of the embedding, which functions as a relaxation of the rank. They end up with the following optimization problem:

$$\begin{aligned} \min_{K,\xi_{ijkl}} \quad & \sum_{(i,j,k,l)\in Q} \xi_{ijkl} + \lambda \mathrm{tr}(K), \\ \text{subject to} \quad & k_{kk} - 2k_{kl} + k_{ll} - k_{ii} + 2k_{ij} - k_{jj} \geq 1 - \xi_{ijkl} \\ & \sum_{ab} k_{ab} = 0, K \succeq 0, \end{aligned}$$

where $K$ is the gram matrix of the embedding, $\xi_{ijkl}$ are the slack variables for the constraints on $K$, and $K \succeq 0$ indicates that $K$ must be positive semidefinite. The actual embedding $X$ can be recovered from the gram matrix by spectral decomposition.

- t-Stochatic Triplet Embedding (t-STE, L. van der Maaten and K. Weinberger, 2012) uses an approach similar to the well-known t-stochastic neighbour embedding (t-SNE, van der Maaten and Hinton, 2008). The authors measure the similarities between points in their embedding using a Student-t kernel with $\alpha$ degrees of freedom:

$$p_i jl = \frac{\left(1 + \frac{\|x_i - x_j\|^2}{\alpha}\right)^{-\frac{\alpha+1}{2}}}{\left(1 + \frac{\|x_i - x_j\|^2}{\alpha}\right)^{-\frac{\alpha+1}{2}} + \left(1 + \frac{\|x_i - x_k\|^2}{\alpha}\right)^{-\frac{\alpha+1}{2}}}.$$

They then maximize the sum of the log probabilities over all triplets $T$:

$$\max_X \sum_{(i,j,k)\in T} \log p_{ijk}$$

using gradient descent. This formulation of the objective not only ensures that the triplet constraints are satisifed: if $(i, j, k) \in T$, the algorithm also decreases the distance between $x_i$ and $x_j$, and increases the distance between $x_i$ and $x_k$.

Other approaches include for example Crowd Kernel Learning (CKL, Tamuz et al., 2011), Fast Ordinal Triplets Embedding (FORTE, Jain et al., 2016) and various others.

It has been proven that if the original points come from the space $\mathbb{R}^m$, one can recover the points (up to scaling and orthogonal transformations) with $O(mn \log(n))$ many triplet comparisons (Jain et al., 2016). On this basis, an ordinal embedding can be used together with more classical machine learning algorithms, such as support vector machines or k-Means, for other machine learning tasks. This approach has for example been demonstrated for classification (Tamuz et al., 2011, Kleindessner and von Luxburg, 2017) or clustering (Kleindessner and von Luxburg, 2017).

### 2.3.2 Other algorithmic approaches

Other algorithmic approaches rely on extracting information from the triplet data directly. These algorithms are hand-crafted for the desired target tasks, such as classification, clustering, et cetera... We collect some example algorithms and briefly describe their approaches.

- Kleindessner and von Luxburg (2017) uses lenses and the lens-depth function, introduced by Liu and Modarres (2011). Assume we have a set of points $D$ equipped with a dissimilarity $d : X \times X \to \mathbb{R}$. For two points $x_i, x_j \in D$, their lens is the intersection of two spheres with radius $d(x_i, x_j)$ centered at $x_i$ and $x_j$. Thus, the lens of two close points has a smaller volume than the lense of two far away points. Using a specialised form of triplets that indicate the most central object out of $i, j, k$, the lens can be used to estimate the proximity of two data points. If, for two points $x_i, x_j$ there are a lot of triplet statements that contain $x_i, x_j$ and another object as the central object, this indicates that their lens must be larger.

  For clustering, the authors build a $k$-relative neighbourhood graph on $D$ by connecting two points $x_i$ and $x_j$ if and only if

  $$V(x_i, x_j) = \frac{N(x_i, x_j)}{M(x_i, x_j)} < \frac{k}{|D| - 2}.$$

  with $N(x_i, x_j)$ being the number of statements that contain both $x_i$ and $x_j$ and have another object $x_k$ as the central object, and $M(x_i, x_j)$ being

the total number of statements that contain both $x_i$ and $x_j$. The obtained $k$-relative neighbourhood graph is then used for clustering together with spectral clustering.

The authors use the insights obtained into lenses together with previous work on lens-depth function to extend the approach also to classification, medoid (most central object) estimation and outlier detection.

- Add Kleindessner and Kernels
- Add ghoshdastidar

# Chapter 3

# Clustering Tangles using Triplet Data

As we described in Section 2.1, the tangles algorithm operates on cuts of data that contain some information about the cluster structure. If we want to cluster a set of triplet data using tangles, we are first faced with the task of processing the triplets to appropriate cuts. In this work, we present two methods for this which we call *landmark cuts* and *majority cuts*. We elaborate on the methods and their motivations in the following sections.

## 3.1 Landmark cuts

In recent years, algorithms have been developed that hope to speed up ordinal embedding by focussing on so-called *landmarks* (Ghosh et al., 2019, Anderton and Aslam, 2019). Landmarks are objects in the dataset for which we know all triplet comparisons. The definition of what constitutes a landmark varies in the literature, but we will use by Haghiri et al. (2019). Assume we have a set of objects $D$, as well as a set of triplet constraints $T$. The triplets have the form $(a, b, c)$, indicating that $d(a, b) < d(a, c)$. In a landmark setting, we have a set of $m$ objects $L \subset D$, for which

$$\forall l_i, l_j \in L \; \forall x \in D : (x, l_i, l_j) \in T \vee (x, l_j, l_i) \in T.$$

If we have landmarks, they make it very easy to define a set of cuts on triplet data: for a combination of landmarks $l_i, l_j$, we can make a cut $P = \{A, \overline{A}\}$ by assigning all points closer to $l_i$ to $A$ and all those closer to $l_j$ to $\overline{A}$. Formally, for two landmark points $l_i, l_j$, we define the set

$$\text{Land}_{ij} = \{x \in D \mid (x, l_i, l_j) \in T\},$$

and call the corresponding cut $P_{ij} = \{\text{Land}_{ij}, \overline{\text{Land}_{ij}}\}$ a *landmark cut*.

For the tangles algorithm, we can relax the requirement on the landmarks, as we do not actually need to know the triplet comparisons between every possible combination of landmarks. Rather, we require that there exists a set of tuples $L' = \{(y, z) \mid y, z \in D\}$ for which:

$$\forall (x, y) \in L' \; \forall x \in D : (x, y, z) \in T \vee (x, z, y) \in T..$$

This will be used in the simulations, as then we can create landmark cuts for tangles by repeatedly picking some objects $y, z \in D$ and sampling all triplet comparisons to all other objects $x \in D$.

Landmark cuts intuitively capture a cluster structure: the closer $x$ is to $l_i$ according to $d$, the more likely it is that $\text{Land}_{ij}$ contains $x$. In the euclidean space, this notion is easily captured: A landmark cut between $l_i, l_j$ is a linear cut between the two points, as illustrated in Figure 3.1.

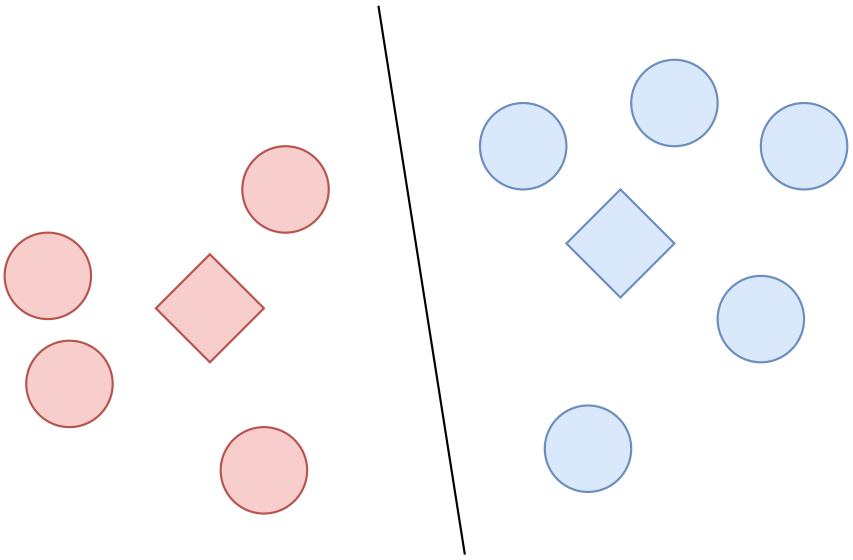


**Figure 3.1:** Example of a landmark cut on a euclidean data set, with the two diamonds being the landmarks $l_1, l_2$. All blue items (left of the line) are contained in $\text{Land}_{1,2}$, all red items in $\overline{\text{Land}_{1,2}}$.

The landmark approach is an unusual way of sampling triplets. In most experiments involving triplets, the triplets are either sampled uniformly at random (Kleindessner and von Luxburg, 2017, Haghiri et al., 2020) or according to a chosen metric, for example maximizing a measure of gained information (Roads and Love, 2021). There is no experimental dataset available which is both sampled according to a landmark approach and exhibits a cluster structure. We thus we rely on simulations for testing landmark cuts. These simulations can be found in Chapter 4.

## 3.2 Majority cuts

As explained in Section 3.1, sampling triplet data in a landmark-fashion is not very widely used in current practice. Due to this, we also present a more general approach to processing triplets to cuts that can be applied to any set of triplets $T$ regardless of the sampling method.

As a motivation, we first think about cuts that are well suited for clustering with tangles. We assume that the objects are clustered based on their similarities to each other, so that similar objects also belong to the same cluster. Based on this, a cut $P = \{A, \overline{A}\}$ works well for clustering if the objects in $A$ are similar to each other and dissimilar to the objects in $\overline{A}$. If we fix a point $a$, one way to generate such a cut is as follows: we center a ball of radius $r$ on $a$ and add all objects contained by the ball to $A$, and all others to $\overline{A}$. This is not perfect, but we know that the objects contained in $A$ have at least some level of closeness (they all have at maximum $2r$ distance from each other).

Next, assume that we have a set of $n$ objects $D$ and a set of triplets $T$ on the objects. We now develop a method of approximating a ball around a point $a$ using the triplet information given. Let

$$L_x = \{t \in T \mid t = (x, b, c), b, c \in D\}.$$

be the set of all available triplets where $x$ is in the left position. Equivalently, we define

$$M_x = \{t \in T \mid t = (a, x, c), a, c \in D\}$$
$$R_x = \{t \in T \mid t = (a, b, x), a, b \in D\},$$

as the sets of triplets where $x$ is in the middle and right position. Using these sets, we define

$$\mathrm{Maj}_a := \{x \in D \mid |L_a \cap M_x| < |L_a \cap R_x|\},$$

as the set of all points that are more often closer to $a$ than they are farther away. We can then define the corresponding cut $P_a := \{\mathrm{Maj}_a, \overline{\mathrm{Maj}_a}\}$, which we refer to as a *majority cut* with *anchor point* $a$.

Assuming we have all triplets, meaning that

$$\forall a, b, c \in D : (a, b, c) \in T \vee (a, c, b) \in T.$$

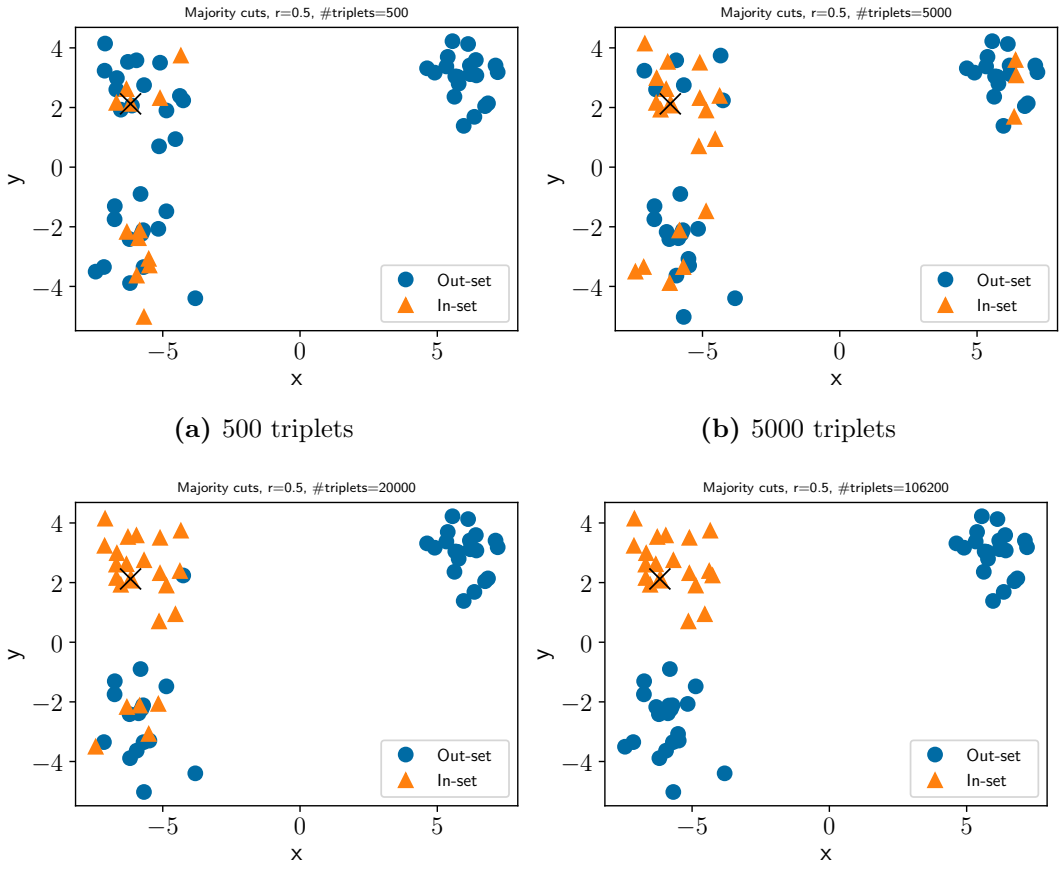
then $\mathrm{Maj}_a$ is a ball around $a$ whose radius is the median distance of $a$ to all other points $x \in D$, as only those points will appear more often closer to $a$ in the triplets than they appear farther away. When $T$ does not contain all possible triplets, we introduce noise on our cuts, and $\mathrm{Maj}_a$ then contains some points outside of a median-sized ball around $a$ as well as it does not contain some points inside the ball.

Majority cuts can be made more flexible by including a ratio $r$ that controls the size of the cuts. We then define

$$\mathrm{Maj}_a(r) := \{x \in D \mid |L_a \cap M_x| < r \cdot |L_a \cap R_x|\}.$$

and call $r$ the *radius* of the cut. $\mathrm{Maj}_a(r)$ is then a ball around $a$ that contains the $n \cdot \frac{r}{r+1}$ points that are closest to $a$ . In a euclidean setting for $r = 0.5$ we thus expect $P_a(0.5)$ to be a ball around $a$ that contains the $\frac{n}{3}$ points that are closest to $a$. To visualize this, we plot a majority cut with a fixed anchor point on a mixture of gaussians in Figure 3.2. As the number of triplets increases, we get closer to a true ball around $a$ containing the $\frac{n}{3}$ closest points. When we have fewer than all triplets available, the majority cut contains noise: some points lie outside of the desired ball, but are included in the cut, and some points lie inside the ball, but are not included in the cut.

What remains is the question of how to choose the radius. We can imagine that if we pick a smaller radius, we will detect smaller clusters, and if we pick a larger radius, we will detect larger clusters. In particular, we should not pick a radius smaller than the smallest cluster we want to detect, else there might exist no tangles on the cuts. On the contrary, we are safe if we pick a radius that is a bit larger, as the tangles algorithm shows good performance on cuts that contain a cluster together with some additional data points (as long as they are not always the same points on all cuts).




**(a)** 500 triplets

**(b)** 5000 triplets

**(c)** 20000 triplets

**(d)** All (106200) triplets

**Figure 3.2:** Majority cuts more closely resemble a ball around their anchor point as the number of sampled triplets increases. We plot $\text{Maj}_a$ with radius $r = 0.5$, generated according to the procedure in Section 3.2. The big X marks the anchor point $a$. The set $\text{Maj}_a$ consists of the orange triangles, which are the points that are twice as often closer to $a$ than they are not (according to the drawn triplets). The blue points are those not in $P_a$.

# Chapter 4

# Simulations

In this chapter and the following one, we explore how tangles perform on triplet data with the methods proposed in Chapter 3. At first, we focus on simulations, as this allows us to control our data precisely. We show in which cases tangles perform well, in which ones they don't and what to keep in mind when applying the algorithm.

We use two different datasets: a mixture of gaussians and a hierarchical block matrix matrix. In both cases, we generate triplets from the data points.

The gaussian setup serves as a first baseline: it is a de-facto standard in clustering and a lot of real-life data is approximately gaussian. We have decided against using more complex data sets such as two-moons, as the focus lies on how the algorithms interact with the generated triplets under various conditions (triplets being corrupted, triplets missing, et cetera...).

The noisy hierarchical block matrix was introduced by Balakrishnan et al. (2011). We use it to illustrate a second property of tangles: in addition to pure clustering, we also produce a hierarchical tree, which can be used for hierarchical clustering.

## 4.1 Terms and methods used

To evaluate the performance of the tangles algorithm, we also need metrics on the performance, as well as other methods to use as a baseline. To compare a clustering against a ground truth, we have to use a scoring function that is independent of the cluster labels. One such function is the *normalized mutual information score* (NMI). For the NMI, 1.0 indicates the same clustering (up to label permutations), 0.0 indicates absolutely no mutual information between the clusterings (such as when our prediction puts all data points in a single cluster). To compare a hierarchy against a ground truth, we use the *average*

*adjusted rand index* (AARI), introduced by Ghoshdastidar et al. (2019). The AARI extends the *adjusted rand index* (ARI), which is a clustering performance measurement similar to the NMI, so that is capable of working with hierarchies. In the AARI, we calculate the ARI over all levels of the hierarchies we want to compare and average over all the obtained scores.

To generate the triplets, we first take the data and calculate a (dis)similarity on it, for example the euclidean distance between two data points. This allows us to determine whether $(a, b, c)$ ($a$ is closer to $b$ than to $c$) or $(a, c, b)$ ($a$ is closer to $b$ than to $c$) holds. Then, we use two different approaches of drawing triplets. The first one is sampling triplets randomly and uniformly from the set of all triplets. The second one is a landmark approach: we fix two randomly sampled points $a, b$, with $a \neq b$ and sample all triplets that have the form $(x, a, b)$ or $(x, b, a)$. For landmark tangles, only the second kind of triplets are useable, as discussed in Chapter 3, thus we will mostly stick to this format.

We also have two possible ways of altering the triplets. First, we can add *noise* to the triplets. If we have a noise level of $p$, then every triplet is flipped with probability $p$, meaning that $(a, b, c)$ would be turned into $(a, c, b)$. We can reduce the total amount of triplets sampled. We will use the term *density* when referring to landmark sampled triplets. If we sample triplets with a density of $d$, that means we sample only a fraction $d$ of all triplets. When explicitly using the number of drawn triplets, we refer to a uniform sampling.

We use multiple baselines to evaluate our tangles algorithms against. First, we use a combination of an ordinal embedding together with a clustering algorithm. This approach also been used in Kleindessner and von Luxburg (2017). It has the advantage of also working with triplet data as opposed to clustering on the original data directly, giving a more fair baseline. Afterwards, we use a standard clustering algorithm for euclidean data on the obtained embedding. There exist numerous algorithms for ordinal embeddings (see Vankadara et al. (2021) for an overview) and for clustering. We use Soft Ordinal Embedding (SOE Terada and Luxburg (2014)), as this has been identified by Vankadara et al. (2021) as one of the top-performing ordinal embedding algorithms for a variety of use cases, which was also the case for us when comparing different baseline algorithms. Additionally, we include T-Stochatic Triplet Embedding (T-STE L. van der Maaten and K. Weinberger (2012)) to have a second baseline. For the clustering algorithm, we use k-Means (Lloyd, 1982), as this is one of the most basic clustering algorithms with usually good performance.

As another baseline we included ComparisonHC (Ghoshdastidar et al., 2019). This, like tangles, is also an algorithm that does not construct an embedding before clustering, and thus might allow for a more fair comparison. Especially in the setting of a mixture of gaussians, the fact that ordinal embedding algorithms aim to reconstruct a euclidean embedding seems to already introduce some bias that might benefit the ordinal embedding algorithms. Nor-

mally, ComparisonHC is a hierachical clustering algorithm, which produces a dendrogram as its output. To use it as a baseline for clustering, we simply cut the dendrogram off such that it produces the desired amount of clusters.

In the experiment figures and discussions, we use the abbreviations L-Tangles for landmark tangles, and M-Tangles for majority tangles.

## 4.2 Gaussian data

### 4.2.1 Experimental setup

We generate a mixture of gaussians as follows: We draw a number of points $n$ each from $k$ different gaussian distributions with means $\mu_1, \mu_2, \dots \mu_k$ and variances $\nu_1, \nu_2, \dots \nu_k$. Each point gets assigned a label $y_j$ that corresponds to the number $i \in \{1, \dots k\}$ of the gaussian distribution it was drawn from. For all of the experiments, unless mentioned otherwise, we use

$$\begin{aligned} n = 20,\ k = 3,& \\ \mu_1 = \begin{bmatrix} -6.0 & 3.0 \end{bmatrix}, \mu_2 = \begin{bmatrix} -6.0 & -3.0 \end{bmatrix}, \mu_3 = \begin{bmatrix} 6.0 & 3.0 \end{bmatrix},& \\ \nu_i = I\ \forall i \in \{1 \dots k\},& \end{aligned}$$

with $I$ as the identity matrix. A sample draw of the data set can be seen in Figure 4.1. We generate the triplets via the euclidean distance between the points, so that the triplet $(a, b, c)$ implies that $\|a - b\|_2 < \|a - c\|_2$. For the Tangles algorithms, we use an agreement of $a = 7$ (around 1/3 the size of the smallest clusters we want to detect, in accordance with Klepper et al. (2020)), and a radius of $r = \frac{1}{3}$, for the majority tangles, such that the cuts roughly have the diameter of the clusters.

All results are the average of 50 runs, where we re-draw both the data points from the gaussian as well as as triplets randomly.

### 4.2.2 Lowering density

First, we want to observe how our methods behave under different numbers of triplets present. As the number of triplet grows on the order of $O(N^3)$ with the total number of points $N$ in the data set, it is only feasible to obtain all triplets for very small datasets. Even with small $N = n \cdot k = 60$, as in our case, there are already 106200 possible triplets. In most settings, the triplets have to be obtained through experiments with real people. If an algorithm performs better with a lower amount of triplets, this can quickly translate into large time, labor and money savings.

To test our algorithm, we draw an increasing amount of triplets in a landmark format from the dataset. We plot the results for a small (60 points) and

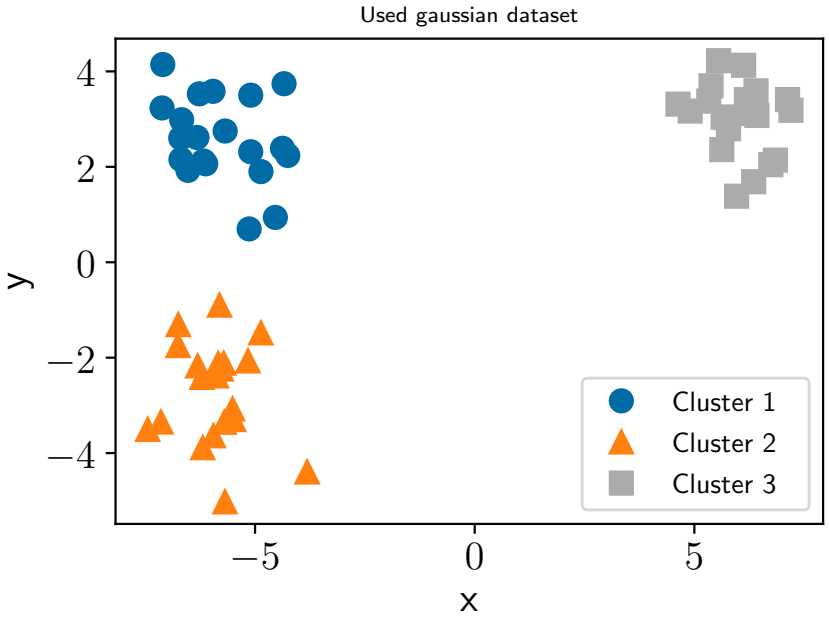
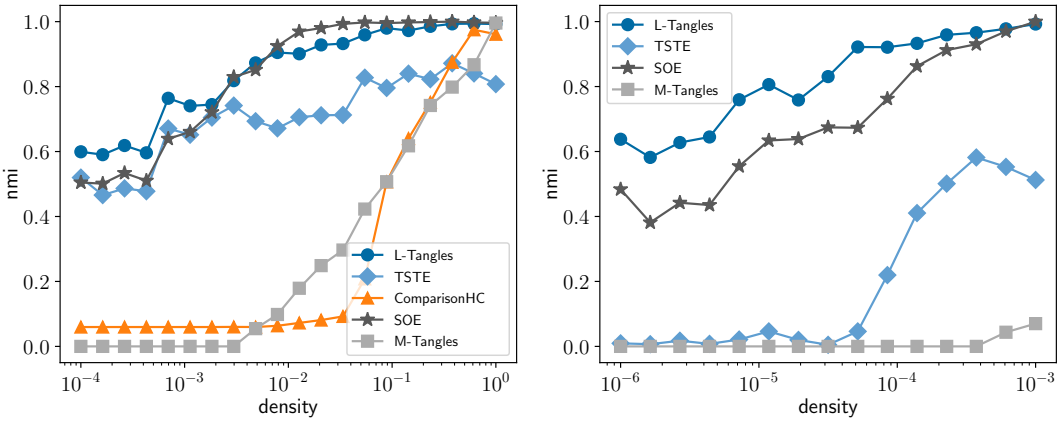



**Figure 4.1:** An example draw of the gaussian mixture used for our experiments.

a large (600 points) dataset in Figure 4.2. Usually, the larger the dataset, the smaller the percentage of all triplets we use, as the number of triplets grows with $O(N^3)$. However, ordinal embedding algorithms empirically perform well with a lot less triplets (for example requiring on the order of $O(nd\log(n))$ for euclidean data (Jain et al., 2016)). Ideally, we would like for the Tangles algorithm to behave similarly.

When looking at the plots, we see that L-Tangles is performing at least as well as SOE, and even significantly better for a wide range of densities in the case of $n = 200$. Tangles thus shows the desired behaviour of requiring a smaller percentage of total triplets with a higher number of data points. As we would have expected, T-STE performs slightly worse than SOE on the small dataset, but interestingly a lot worse for the larger dataset. ComparisonHC and M-Tangles perform about the same level, but both stay far behind L-Tangles and SOE. With the larger dataset, we could not test ComparisonHC, as our implementation requires constructing a $n^4$ matrix during the training step (this could possibly be remedied using a different implementation).




**(a)** 20 points per cluster **(b)** 200 points per cluster

**Figure 4.2:** Influence of the triplet density on the performance. We plot the NMI of different clustering methods against the percentage of the triplets generated from a draw of a gaussian mixture with 3 clusters. The triplets are generated in a landmark format. We draw 20 data points from each cluster for the left plot, and 200 data points for the right plot. On the x-axis we have the density, where a density of 0.1 means that we only use 10% of the total number of triplets. The embedding methods (SOE, T-STE) are followed by k-Means. The Tangles methods (L-Tangles, M-Tangles), are applied with $a = 7$ for $n = 20$ and $a = 70$ for $n = 200$. ComparisonHC was left out of the right plot due to computational issues.

### 4.2.3 Adding noise

Next, we want to observe how our algorithm behaves under added noise. This is an important model: most applications of triplet data use triplets that are generated from real humans. They might disagree on which objects are closer and which are not, which can be modelled as noise on the responses of our triplets. The higher the noise, the more disagreement there is about the similarities of objects, so it would matter more about which person you ask than which objects you present them.

We plot the performance of our algorithms over increasing noise in Figure 4.3, with all triplets sampled (density 1.0). We observe that both landmark and majority tangles fall off more quickly in performance with increasing noise than the other algorithms, with L-Tangles still being better than M-Tangles. We observe however, that until noise levels of 0.1, all algorithms perform the same, meaning that L-Tangles can still perform well with low to medium levels of noise. Interestingly, ComparisonHC, which required a lot of triplets to achieve acceptable performance, seems very noise resistant, performing about as well as SOE until noise levels of 0.3.

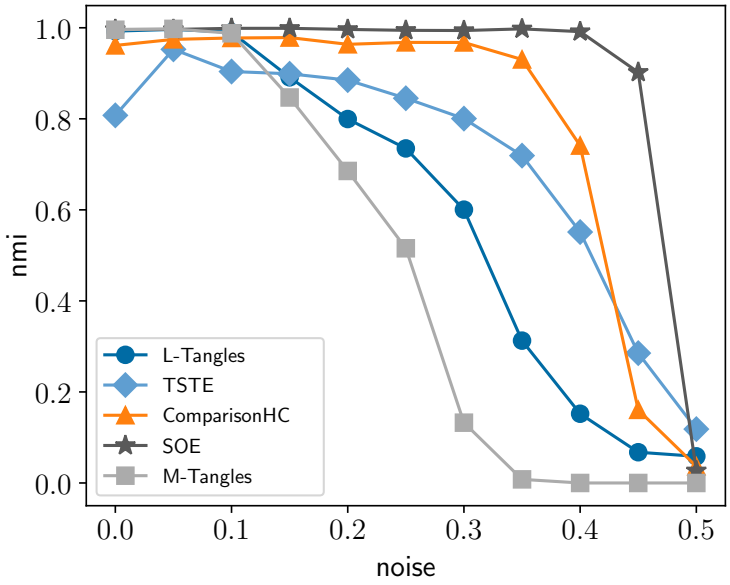
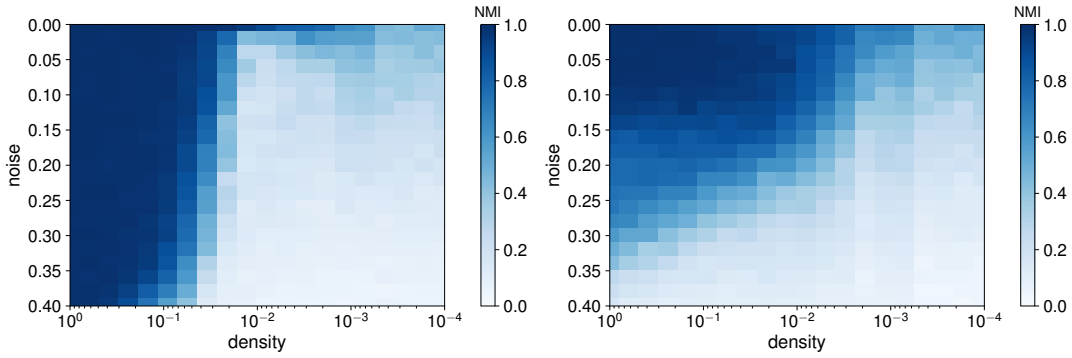



**Figure 4.3:** Influence of the triplet noise on the performance. We plot the NMI of our chosen clustering methods against the noise that we introduce on the triplets. We use 3 clusters and 20 data points per cluster and sample all possible triplets. A noise level of 0.1 means that we flip 10% of the triplets (turning for example $(a, b, c)$ to $(a, c, b)$).

### 4.2.4 Adding noise and lowering density

In this experiment, we will merge Subsection 4.2.2 and Subsection 4.2.3. We have seen that L-Tangles performs well with a low amount of triplets (a bit better than SOE) and has reasonable performance for noisy triplets (albeit worse than SOE for high noise). We are now interested to see how large the area is where L-Tangles can still outperform SOE when we consider both a low density and noisy triplets, as these two factors are probably the most interesting variables when choosing an algorithm to evaluate triplet data.

We generate two heat maps in Figure 4.4. There, we can see that L-Tangles outperforms SOE in quite a large region in the low-noise, low-density regime. We would imagine this effect to be even larger for a $n = 200$ setup, but found this computationally too intensive. On the contrary, SOE performs better in the high-density, high-noise regions, with about similar performance for the cases of low-noise high-density (perfect clustering) and high-noise low-density (random clustering).




**(a)** SOE **(b)** L-Tangles

**Figure 4.4:** L-Tangles shows a performance advantage over SOE in the low-density, low-noise regime. We plot a heatmap of the NMI of SOE and L-Tangles over noise (y-axis) and density (x-axis) of our the triplets, generated from a mixture of gaussians with 3 clusters and 20 data points per cluster. The triplets are drawn in a landmark format. The regions with a darker shade of blue indicate better performance of the algorithm. Note that the noise increases as we move down, and the density decreases as we move right. This means that clustering gets harder when moving right and/or down and easier when moving left and/or up.

### 4.2.5 Missing triplets and imputations

In this experiment, we use the small gaussian data set and gradually sample an increasing number of triplets for it. This is very similar to the setup in Subsection 4.2.2, but this time we sample triplets uniformly at random without replacement from the set of all triplets. In this setup, we are not able to use L-Tangles as is, as there will be missing values in the cuts.

An idea would be to impute the missing triplets for the cuts. For, this we just build up our landmark cuts as we did before, but we mark entries for which we don't have triplet information. These missing values can then be imputed via different methods. We have used *random*, *k-nearest neighbour* ($k$-NN) and *mean* imputation. Random simply sets all missing values to 0 or 1 with equal probability, $k$-NN imputes a missing entry in a landmark cut with the value that the most similar cut has in that position (closeness being calculated via the manhattan distance), and mean imputes the value with the mean of all other cuts in that position. We show the results for different imputation methods in Figure 4.5. One can see that $k$-NN performs quite reasonably, and the simplest choice of $k = 1$ even achieves the best performance.

We compare the performance of our different algorithms versus the number of triplets drawn in Figure 4.6, where L-Tangles was imputed with 1-NN. Here, L-Tangles loses out against SOE, which achieves an acceptable level of

performance at much lower amounts of triplets. However, we can see that L-Tangles is competitive against M-Tangles and ComparisonHC. T-STE also reaches an acceptable performance earlier, but saturates quicker at an NMI of 0.8, compared to the other methods, which achieve an NMI of $> 0.9$ with $> 10000$ triplets

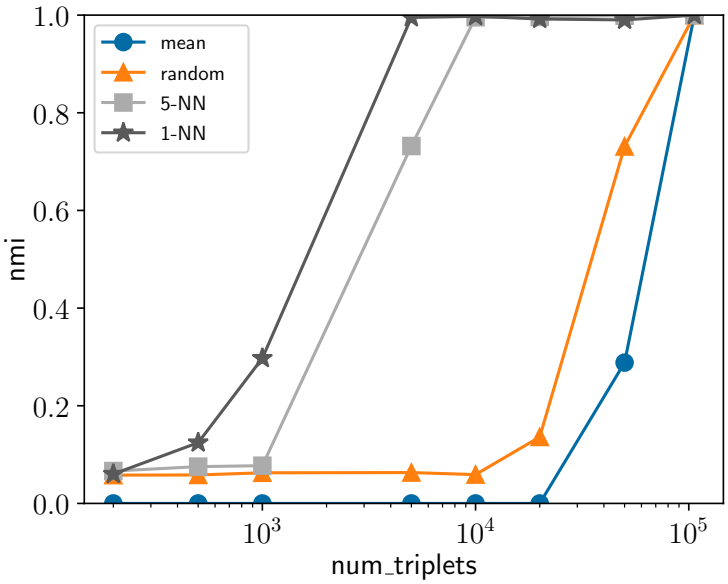



**Figure 4.5:** 1-NN performs well for imputing missing triplets for L-Tangles. We plot the performance of L-Tangles over the number of triplets available for different imputation methods over our small gaussian dataset with 3 clusters and $n = 20$. We use different imputation methods to fill in the missing values and then cluster the resulting cuts with L-Tangles. The *k-NN* methods use k-nearest neighbour imputation, *random* assigns 0 or 1 to the missing values with equal probability, and *mean* assigns the missing values the mean value over all other cuts at that position.

## 4.3 Hierarchical data

### 4.3.1 Experimental setup

We generate a noise hierarchical block matrix (Balakrishnan et al., 2011). The hierarchy described by this model has the form of a complete binary tree, and the similarities of the data points are described via a similarity matrix $M$, where the entry $M_{i,j}$ describes the similarity between the points $i$, $j$. In this matrix, the elements in the same cluster have the highest similarity $\mu_0$, and for each level $l$ that two classes are removed from each other in the hierarchy, their similarity decreases by $\delta$. We can also correupt the similarity matrix by

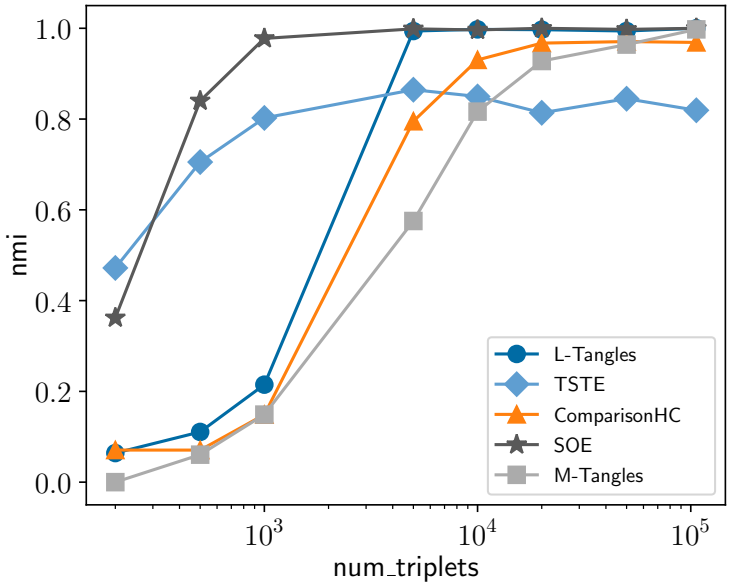



**Figure 4.6:** Influence of missing triplets, when triplets are not drawn in a landmark format but uniformly at random. Analog to Figure 4.5, but this time we plot the performance of various other evaluation methods on the small gaussian data set against L-Tangles. We impute the missing values in the L-Tangles algorithm with 1-NN.

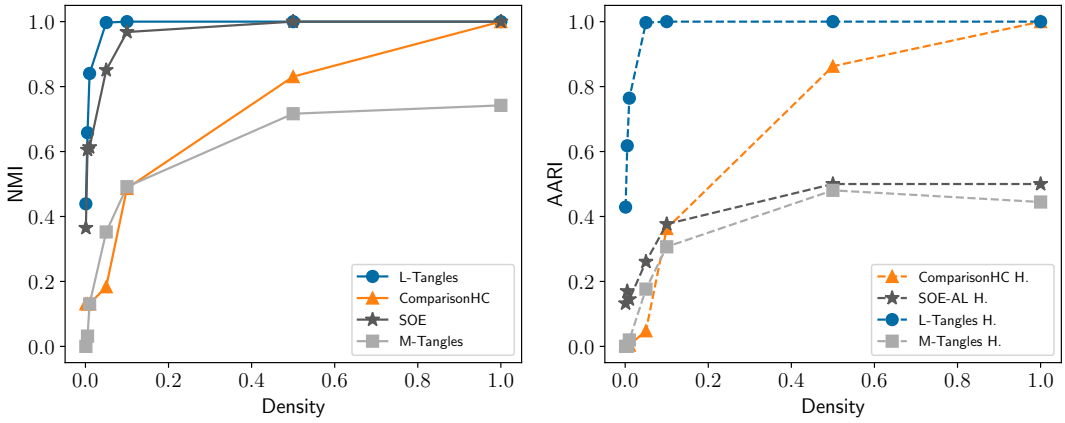
an added noise matrix $R$. We then receive the noisy hierarchical block matrix $M' = M + R$. In our setup, we use a noise matrix $R$ where every entry is simply drawn from a normal distribution with mean 0 and standard deviation $\sigma$. More about the generation process can be read in Ghoshdastidar et al. (2019).

We choose a relatively simple setup of 4 clusters with 10 data points each, an initial class similarity $\mu_0 = 5$ and a similarity decrease of $\delta = 1$. In this setup, there are two kinds of noise we encounter: the noise that is injected into the hierarchy itself via the noise matrix and the noise that is added to the triplets. The triplet noise has the same form as in Section 4.2: on noise $p$, every triplet has a chance of $p$ percent to be flipped. We investigate how our algorithms perform under both noise models, as well as under a lowered density. When evaluating, we compare both the final clustering (the lowest level of the hierarchical block matrix) as well as the obtained hierarchy to each other. To produce a hierarchy with SOE, we have applied an agglomerative clustering algorithm with average linkage (AL) on the obtained embedding. As in the gaussian setting, all results are the average of 50 runs, where we re-draw both the data points as well as the triplets randomly.

### 4.3.2 Lowering density

As in Subsection 4.2.2, we investigate how our algorithms perform under a lowered density. We draw the triplets in a landmark-format. The results can be seen in Figure 4.7.

We can see that L-Tangles performs about on-par with SOE in the final clustering case (not taking the hierarchy into account), and greatly outperforms all the other algorithms, while M-Tangles and ComparisonHC perform about equally well, wtih ComparisonHC getting better results in the high triplet case.

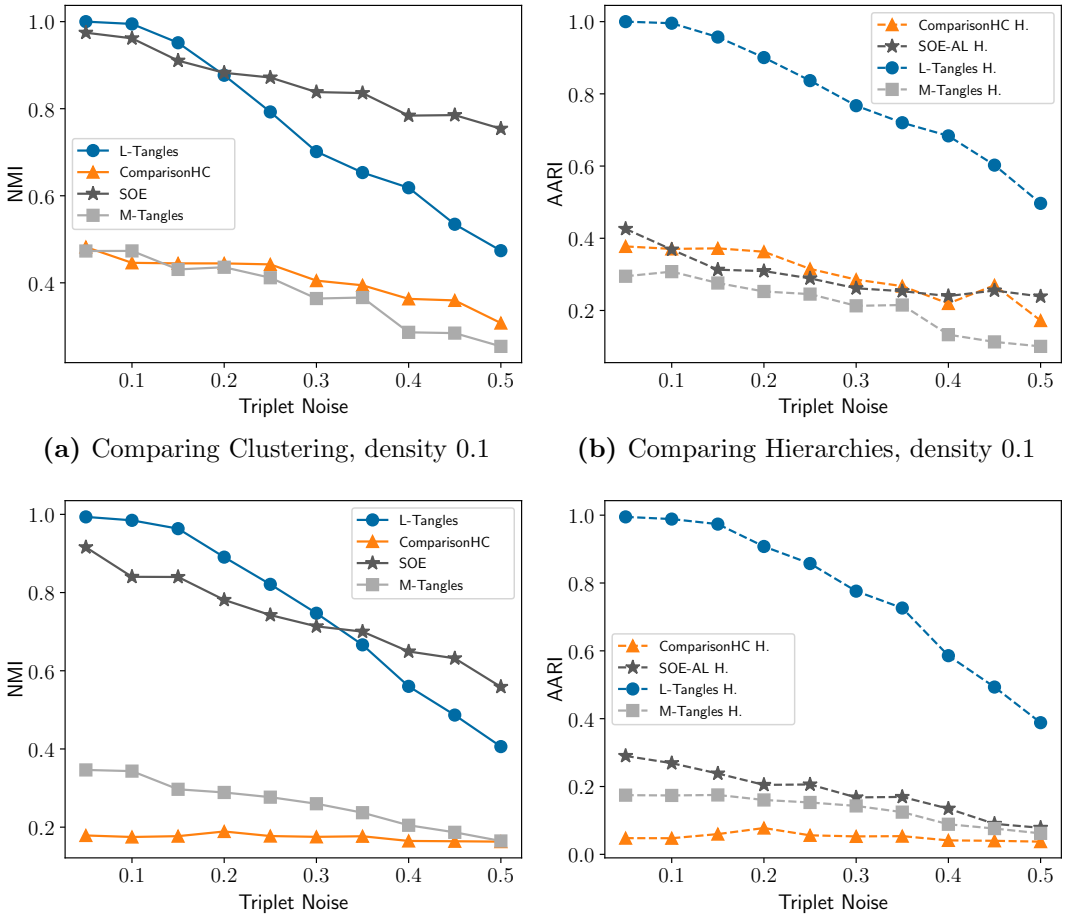



**(a)** Comparing Clustering **(b)** Comparing Hierarchies

**Figure 4.7:** Performance of our algorithms against the density of the triplets drawn. We draw the triplets in a landmark format and they are generated from a hierarchical noise matrix with 4 clusters and 10 data points each. In the left, we see the NMI of the obtained clusterings versus the ground truth, where we only use the final clustering to evaluate. The ordinal embedding algorithms (T-STE, SOE) have been followed by k-Means. On the right, we plot the AARI of our methods against the density, where we take the whole hierarchy into account. To obtain a hierarchy for SOE we have applied agglomerative clustering with average linkage to the obtained embedding instead of k-Means.

### 4.3.3 Adding triplet noise

Similar to the gaussian setup in Subsection 4.2.3, we increase the noise on the sampled triplets and evaluate the performance of our algorithms. The results can be seen in Figure 4.8. We repeat the setup under two different densities, with 10% and with 5%, to see if the density has an influence on the noise suspectibility of the algorithms.




**(a)** Comparing Clustering, density 0.1

**(b)** Comparing Hierarchies, density 0.1

**(c)** Comparing Clustering, density 0.05

**(d)** Comparing Hierarchies, density 0.05

**Figure 4.8:** Performance of our algorithms against the noise introduced on the triplets. If we have a noise of 0.1, we flip a triplet with probability 10%. We again use a hierarchical block matrix with 4 clusters and 10 points each. On the top row, we draw all triplets in a landmark format, on the bottom row we drawn 10% of them. We draw 20 data points from each cluster in the left column, and 200 data points in the right one.

First, we look at the pure clustering. In the 10% density case, SOE outperforms all other methods unless the noise is low ($< 0.2$), where L-Tangles performs better. If the density decreases (5%), L-Tangles keeps the advantage until higher noise levels (to about 0.35). This is similar to what we see in the gaussian setup. L-Tangles and SOE however perform a lot better than ComparisonHC and M-Tangles in for both densities and all noise levels. For the hierarchy, L-Tangles greatly outperforms all other methods for both densities, with the other methods performing on roughly the same level. Interestingly, we see that for both comparing hierarchies as well as clustering, ComparisonHC seems very dependent on the number of triplets present, as the NMI on all

noise levels roughly doubles when we go from 5% to 10% density.

### 4.3.4 Adding hierarchy noise

This setup differs from the experiments done on the gaussian data. Here, we vary the noise that we add directly to the hierarchical block matrix $M' = M + R$. $R$ is a matrix whose entries all consist of gaussian noise that is drawn from a normal distribution with mean 0 and variance $\sigma^2$. We set $\sigma^2$ to various values and evaluate the performance of our algorithms. The result can be seen in Figure 4.11. In the normal clustering case, SOE performs the best over the board, with L-Tangles falling off pretty sharply on the introduction of noise into the hierarchy, but still outperforming M-Tangles and ComparisonHC. In the hierarchical case, L-Tangles outperforms all other methods until hierarchy noise of 2, whereafter SOE and L-Tangles achieve similar performance. We assume that this is because the similarity difference between two completely unrelated classes is at most 2. After a noise level of 2, we thus get into a regime where the noise simply overpowers the information left in the similarity matrix.

We can see an interesting effect here: L-Tangles shows a steep decline in performance after the introduction of even a small amount of noise, which is not present in our other noise model, where the noise was added to the triplets directly (see Subsection 4.3.3). In the other noise model, we see a much smoother decline in performance with added noise on the triplets. In fact, even when adding a vanishingly small amount of noise (say $10^{-6}$) we see the same steep decline in performance. This illustrate a potential shortcoming of the tangles algorithm, which we will now elaborate on.

First, we want to get some intuition about the hierarchical model. As a visualisation aid, We have plot a representation of the hierarchical model in Figure 4.9. Care should be taken: this does not accurately visualise the distances between the points in any way, but is merely a useful tool to illustrate cuts on the data. We now look at the landmark cuts that we retrieve under the two noise models, with a noise of 0.1 in the triplet noise case and vanishingly small noise $\epsilon = 10^{-6}$ in the hierarchy noise case.

Assume that we select two points, one from one of the left clusters, and one from one of the right clusters. When we generate the landmark cut that is associated by those two points (by putting all points that are closer to the left point in the left set of the cut, and putting all points that are closer to the the right point in the right set) we receive is a *coarse* cut, that divides a higher level of the hierachy (roughly into left and right). As we see in Figure 4.10a and Figure 4.10b, this cut looks pretty similar under both noise models. In the triplet noise case, we can see that some of the data points have been assigned to the wrong cluster, but this is expected.

Next, we will take two points, $a$ from the bottom-left, $b$ from the top-

left cluster. The resulting landmark cut is a *fine cut*, that separates between lower levels of the hierarchy. In this setting, something interesting happens in the hierarchical noise model: the points from the right clusters get randomly assigned to left-set or right-set. To understand this, let's first look at what happens when we have no hierarchy noise. In the hierarchical block model, all distances only depend on how far removed the points are in the hierarchy, thus the bottom-left and top-left clusters are correctly separated by the cut. All points from the bottom-left cluster will be in the left set, all cuts from the top-left cluster will in the right set. Let now $c$ be a point from the bottom-right or top-right cluster. Then, $d(c, a) = d(c, b)$. If we don't have noise on the hierarchy, this does not pose a problem: in our landmark cut implementation, we decided to break ties deterministically, ruling $c$ is in the $\text{Land}_{ab}$ only if $d(c, a) < d(c, b)$. Thus, $c$ will not be contained in $(a, b)$. As a result, the set $\text{Land}_{ab}$ will contain only the bottom-left cluster and no other points. However, when we add even the tiniest amount of noise to the hierarchical model, then for all points $c'$ from on of the right clusters it will randomly be either $d(c, a) > d(c, b)$ or $d(c, b) > d(c, a)$. This means that those points will be randomly assigned to $\text{Land}_{ab}$ (or not). This can also be seen in Figure 4.10c. For the triplet noise case, we have the ideal assignment (all points from the right clusters are assigned to the right set), but some points are again randomly assigned wrongly, see Figure 4.10d.

Now, how does that influence our clustering? Let us step through an example clustering that L-Tangles would make with a hierarchical noise model. At first, tangles would receive the coarse cuts (they are cheaper as they are more balanced and more frequently present) and subdivide the points into the left and the right clusters. Next, at some point, we would need to orient one of the fine cuts. This will subdivide either the left or right clusters (depending on which cut we receive). but the random assignment in the other cluster (that one which is not subdivided) prevents us from orienting the fine cut of the other clusters consistently (depending on agreement, but if we have to orient two fine cuts of the same cluster after another, even a very small agreement will not allow consistent orientation). Thus, we will end up with three clusters, the two clusters that are subdivided by the first fine cut that appeared, and the other two clusters merged to one.

On the other hand, if we only deal with (low) triplet noise, we can orient the first coarse cut in any way we want, and the fine cut then gets orient in one direction in the left subtree and in the other direction in the other subtree. We end up with the correct amount of clusters, and the missclassifications will only be a few random points that are assigned to a wrong cluster due to triplet noise.

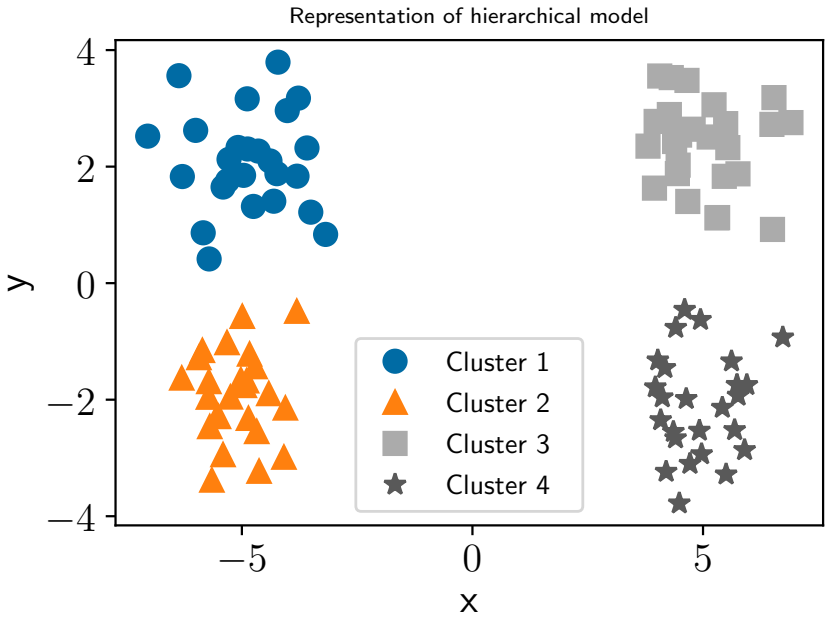
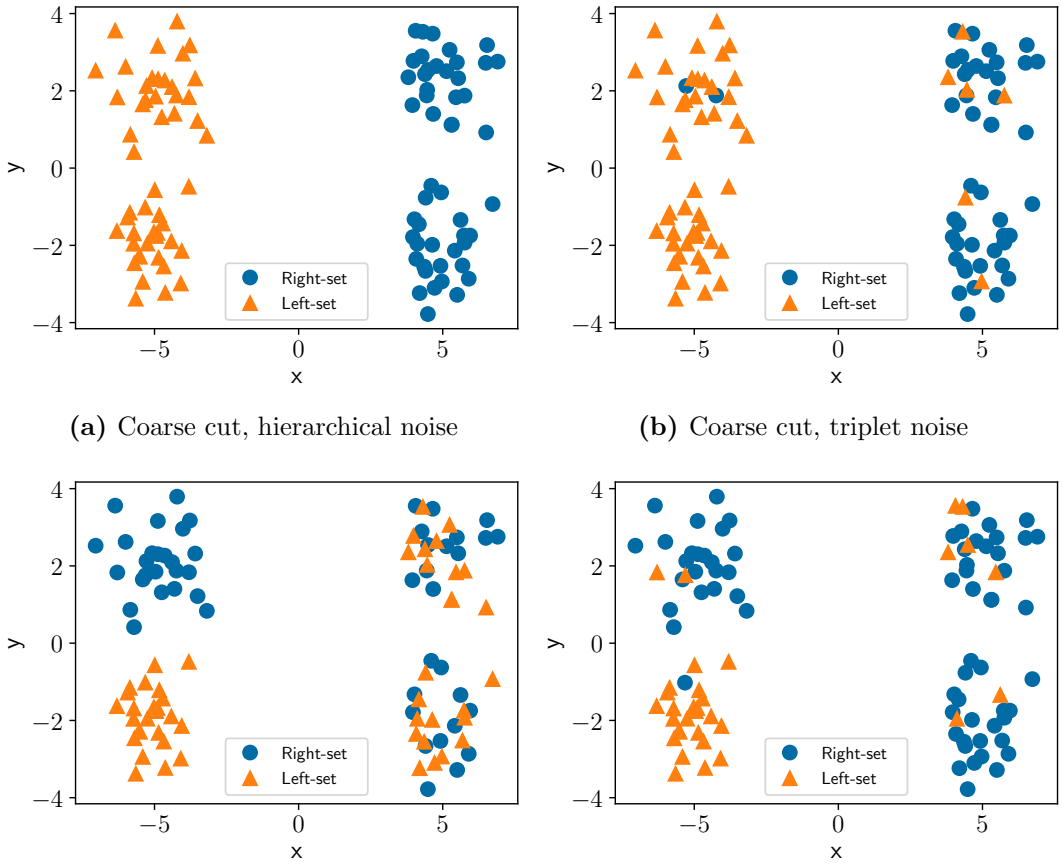



**Figure 4.9:** A euclidean representation of the hierarchical model with a few more data points drawn. We can see a sort-of hierarchy between the clusters, where the left and right side are two far removed clusters, which can be subdivided in bottom-left, top-left and bottom-right, top-right. Keep in mind that this representation is only a visualisation aid and does not accurately reflect the actual similarity between data points.



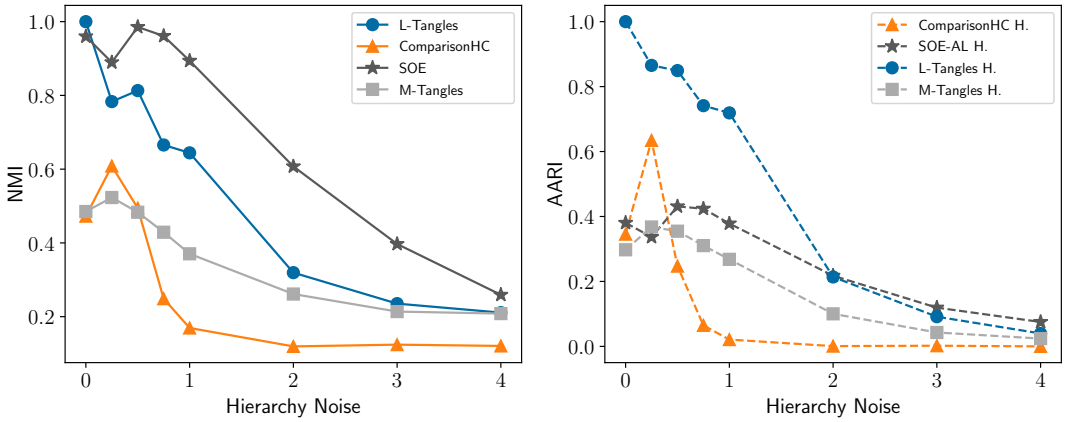

**(a)** Coarse cut, hierarchical noise

**(b)** Coarse cut, triplet noise

**(c)** Fine cut, hierarchical noise

**(d)** Fine cut, triplet noise

**Figure 4.10:** Visualisations of the cuts we receive under both noise models of the hierarchical setting, analog to Figure 4.9. Hierarchical noise means noise that we add directly to the hierarchical similarity matrix, while triplet noise means the percents of triplets answered wrongly. We assume landmark cuts. The coarse cuts are those that separated higher levels of the hierarchy (so left and right clusters), while the fine cuts further subdivide left and right into bottom-left, top-left, bottom-right and top-right.




**(a)** Comparing Clustering **(b)** Comparing Hierarchies

**Figure 4.11:** We plot the performance of our algorithms against the noise on the hierarchical block matrix. A noise of 1 means that the each entry in the similarity matrix of the hierarchies is independently corruped with additive gaussian noise $r_{ij} \sim \mathcal{N}(0, 1)$. We again use a hierarchical block matrix with 4 clusters and 20 data points. On the left, we report the NMI of the final clustering against the ground truth. On the right, we also take the hierarchies into account, reporting the AARI.

### 4.3.5 Discussion

In this section, we have reported the performance of the Tangles algorithm on two synthetic data sets, gaussian and hierarchical data. We could see that with landmark triplets, L-Tangles consistently had the best or among the best performance, even outperforming SOE in regimes of low noise. Additionally, L-Tangles proved to be the best performing algorithm when trying to determine the hierarchy in our hierarchical model. If we are not presented with landmark triplets, we could observe that L-Tangles is still capable of reaching an acceptable performance (a bit better than ComparisonHC, which is a state-of-the-art clustering method for clustering triplets without creating an intermediate representation). Interestingly, we found ComparisonHC to be very reliant on a having a large amount of triplets available to have an acceptable performance, being outperformed by L-Tangles in almost all settings (besides in the case of high noise and almost all triplets available).

Overall, M-Tangles did fare worse than L-Tangles, but it still had acceptable performance, that was overall comparable to ComparisonHC. However, M-Tangles introduces another hyperparameter, the radius, which needs to be tuned. Thus, if triplets are present in landmark format, we strongly prefer L-Tangles. Even if the triplets are not in landmark format, imputing the triplet responses with a simple 1-NN method and then using L-Tangles seems pre-

ferrable to using M-Tangles. We have still included M-Tangles because we think it can serve as an interesting baseline from which to build more sophisticated methods, for example one could think about weighing the triplets in a certain way when counting them up (if we have two points $A$, $B$ and we already know they are very close, then $(A, C, B)$ is a strong indicator of $C$ and $A$ being in the same cluster).

We have also shown some cases where the tangles algorithms performs subpar: for example in the case of high noise or a large amount of missing triplets (non-landmark format). We have also raised some issue with the noise in the hierarchical model, and we now want to discuss whether this points to an artifact in the model or a more serious flaw in the tangles algorithm.

In Subsection 4.3.4, we have used two different noise models: adding noise to the triplets and adding noise to the hierarchical block model. This points to two fundamentally different assumptions about our data. Let's think of our data as a hierarchy of 2 clusters, which separate again into 2 clusters. The clusters are fruits (apples, bananas) and vegetables (zucchini and potatoes). If we wanted to cluster this data using triplet data, we would gather a lot of people, present them with three images of our objects, and ask them whether the first image is more similar to the second or third one. In the triplet noise model, we assume that either some people answer "wrongly", or that some objects might actually be more similar to an object from another cluster than to ones from their own cluster (maybe we have an image of a banana and an image of a zucchini that have both been shot in front of a beach and thus look similar). Most of the time however, there is some kind of hidden way to compare items from entirely unrelated hierarchies. In general, apples might always be more similar to potatoes than to zucchinis for example. As a consequence, if we for example have a landmark cut from an apple and a banana, it would contain all the apples and all the potatoes.

In the case of hierarchy noise, we have some objects that are flat out more similar to one category than another. For example, in the set of apple images, we might have a lot of green apples, which are almost always more similar to zucchinis than to potatoes, and a lot of yellow apples, for which it is the other way around. For this reason, when we look at a landmark cut of a zucchini, this one might contain all bananas and all green apples. As explained in Subsection 4.3.4, clustering this poses a problem for the tangles algorithm. The actual amount of noise added doesn't matter (as long as it is smaller than the similarity decreases between distances), as this information gets lost when turning the similarity matrix into triplets.

Overall, which of the two noise models sounds more realistic might depend on the problem setup, how we present the items, how we ask the triplet questions et cetera. Nontheless, even in the hierarchical noise model, L-Tangles has a very good performance, meaning it can be used without thinking too much

about which noise model we are confronted with.

# Chapter 5

# Case study: data from psychophysics

In this chapter, we use the tangles algorithm for real world data. The chapter aims to show both the viability of tangles as a data evaluation tool, as well as to serve as an instructive application for how a researcher might use tangles for their own experiments. We use the data from the thesis of Schönmann (2021) and will complement some of their analysis using tangles.

## 5.1 Data background

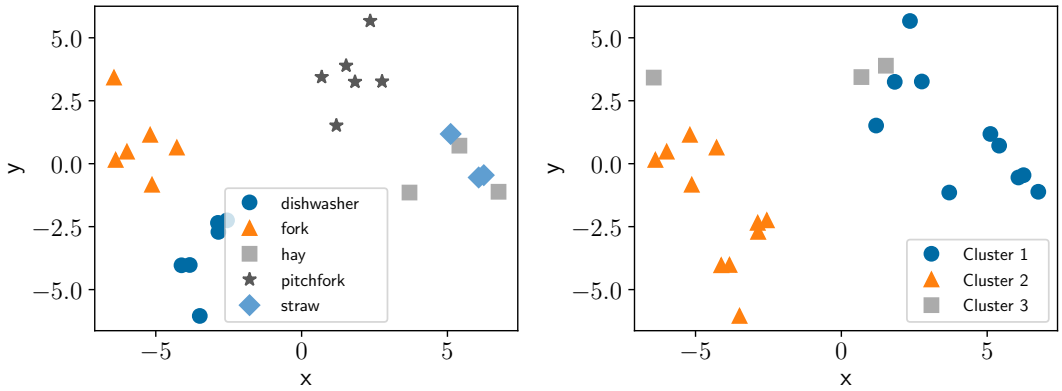
The data we use was collected by Schönmann in her bachelor thesis. Schönmann originally constructed the dataset to investigate how the formulation of a triplet question influences the perception of a person. It consists of the triplet data obtained from multiple participants using different triplet questions and image sets. To collect the data, the observers were presented with three images randomly drawn from the a specific set of images, and together with one out of four different questions. The *odd* question, which we analyse in our setting is phrased as: *Which is the odd one out?*. As explained above, if out of $a, b, c$, $a$ is the odd one out, we transform this into two triplets $(b, c, a)$ and $(c, b, a)$. Out of the three image sets, we will study the so-called thematic image set more closely. It consists of 5 classes which can be roughly divided into two themes: barn (straw, hay, pitchforks) and kitchen (forks, dishwashers).

For each observer, image set, and question combination, the data set consists of 462 unique triplets. Observer 2 has repeated the experiment again over a month later, but we have decided not to include those triplets to stay faithful to the original evaluation.

## 5.2 Applying Tangles

In the following, we make an example of how to apply tangles to the obtained triplet data. As there is a lot of data present, it is not feasible to repeat our evaluations for all possible data points. We choose one particular example to step through rigorously, and then we compare it to one other example. For our evaluations, we use data points with the odd-one out triplets, as these have been reported in the work by Schönmann as having a higher embedding accuracy. We also use the thematic image set, as Schönmann has reported embeddings that can be cleanly separated into different categories.

We start out with observer 2. We first reproduce the results by Schönmann by embedding the data with SOE into two dimensions (see Figure 5.1a). One can linearly separate two sets of clusters, which correspond to a divide between kitchen objects (dish-washer, fork) and barn objects (hay, straw, pitch fork). We then apply majority tangles with an agreement of 3 and a radius of 1 (a lower radius produced only or two clusters) to the triplets and obtained a clustering. These cluster labels are visualised in Figure 5.1b) using the previously obtained embedding. Care must be taken: the embedding from SOE is not a ground-truth embedding and just serves as a visualisation aid. If the elements of a cluster are far apart in this visualisation, this does not mean that the clustering is wrong – it could just as well be that the embedding is wrong or simply cannot capture the cluster structure appropriately. This is similar to how T-SNE can produce very deceptive embeddings of data.




**(a)** Original classes **(b)** Tangles clusters

**Figure 5.1:** Embedding of the odd-one-out triplet data from observer 2 on the thematic image set. On the left, we see the original classes, on the right we see the predictions that we receive by applying majority tangles with an agreement of 3 and a radius of 1.

We can see that the clustering from tangles also produces a similar divide between kitchen (orange triangles) and barn (blue circles) items. However, we see a third cluster structure (grey squares), which is a mix between two pitch forks and a kitchen fork. When we look at these items (depicted in Figure 5.2a), we can notice that they look more dissimilar to their counterparts in the kitchen or barn cluster. The fork (left in Figure 5.2a) has a design that is more reminiscent of pitch forks, and the two pitch forks look more clean than the other pitchforks in Figure 5.2c (no dirt on them, not depicted lying in grass). Thus, it makes sense that these three items are judged as more similar to each than their counterparts in the kitchen and barn cluster and thus get put into a separate clustering. Interestingly, that is an insight that could not be reached from the SOE embedding alone and might provide valuable information for a researcher.

Next, we want to see how one might use the hierarchy reconstruction and the explanatory power that tangles provides. For this, we first plot the hierarchy that we receive from the tangles algorithm in Figure 5.3. To get a better idea of how the hierarchies look like, we plot the soft clustering at each node, which can be seen in Figure 5.4. Here, each cut corresponds to one node of the hierachy and we can make out which clusters belong to which nodes. As expected, we first see a coarse, thematic divide between kitchen and barn (nodes 18T, 0F, see Figure 5.4a), with the clean-looking pitchforks being placed together with the kitchen cluster. We then see a finer clustering of the kitchen cluster (node 0F), which is split into a set of various kitchen items (node 19F), and the cluster of special forks (node 4F). The fact that the special forks belong to the coarse kitchen cluster and only get separated off in a later step could be interpreted as the participant thinking of the clean-looking pitch forks as thematically belonging more into a kitchen than a barn. This is an insight that we couldn't have gotten from the ordinal embedding alone, highlighting a possible strength of tangles.

**(a)** Third cluster of different forks (gray squares)

**(b)** Other kitchen forks

**(c)** Other pitch forks

**Figure 5.2:** The images of all forks that are present in the thematic data set. In a), we have put the pitch forks and forks that landed in the gray squares cluster in the tangles clustering (see Figure 5.1), which contained a mixture of kitchen and barn items. In b), we have depicted all other kitchen forks (all contained in the orange triangle cluster associated with kitchen items) and in c) we have depicted all other pitch forks (contained in the blue circle cluster associated with barn items).

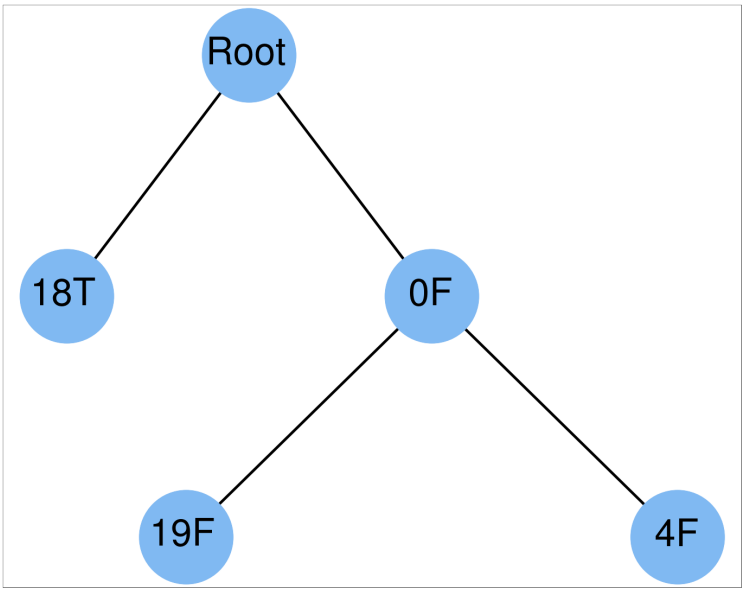


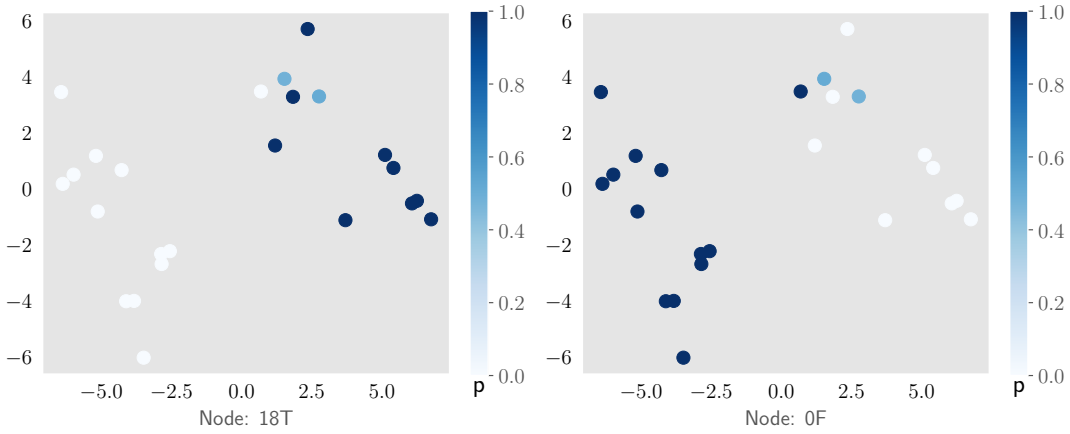

**Figure 5.3:** The hierarchy produced by the tangles algorithm on the thematic image set. The hierarchy starts at the root and each other node represents a splitting cut, which separates the data further. The label of the node is the number of the cut that caused the split to happen, together with the orientation (T left orientation, F means right orientation). This label can be used to identify the node in later processing steps.

The last step showcased the strength of the hierarchical clustering of tangles. Next, we show how we can use tangles to explain the clustering: why does a particular image belong to the kitchen or barn cluster? For this, we can look at the characterizing cuts of the clusters. As a reminder, if we look at a certain splitting node, its characterizing cuts are those that are always oriented in the same direction in the left subtree and in the other direction in the right subtree.

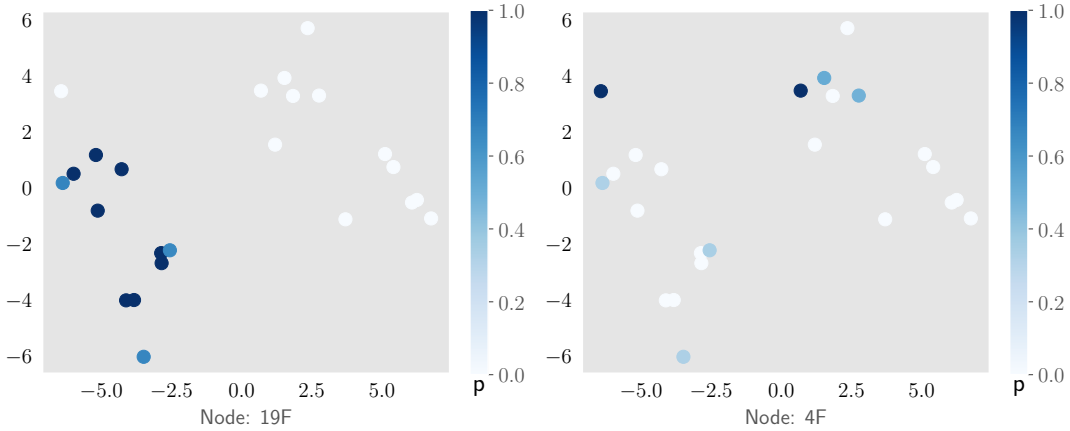
We have visualised the characterising cuts in Figure 5.5, together with the images that induced the particular cuts. As our cuts are interpretable (a majority cut with anchor point $a$ contains points that are close to $a$), we can directly use this interpretation for our clustering. As we can see in Figure 5.5a, the items that are in the barn cluster have landed there because they are similar to the two straw/hay images that we have plotted in Figure 5.5b. This is similar to our intuition, straw and hay intuitively belong in a barn setting and definitely not in a kitchen, while the pitchforks might be a bit more ambiguous.

Our interpretation is that the items that have landed in the cluster of other kitchen items are there because they are similar to the dishwashers we have plotted in Figure 5.5d. This also makes sense, as we would interpret the dishwashers to very clearly belong into a kitchen environment, while the forks might be more ambiguous.

Overall, there is some small caveat to our explanation: In a majority cut, we only have a satisfying explanation in one direction: We know that if a point $b$ is in the majority cut that has anchor point $a$, then $b$ is close to $a$. However, the reverse direction might be a bit unsatisfying: if a point $c$ is not in the majority cut, we know that it is not close to $a$. If we want to know how the cluster of forks and pitchforks formed, saying that they are dissimilar to the dishwashers plotted in Figure 5.5d is not a really strong argument. We for example rather know that they are similar to a certain other item. To remedy this, we can use more interpretable cuts. If we would have sampled the data in a landmark format, we would have been able to make statements of the form: $a$ is in a certain cluster because it is closer to $b$ than to $c$.



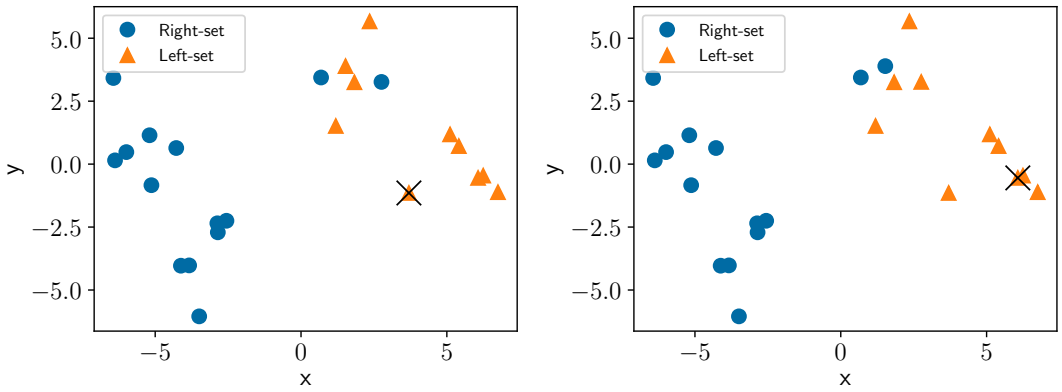

**(a)** Coarse cluster (barn - kitchen)




**(b)** Fine cluster (kitchen items - special forks)

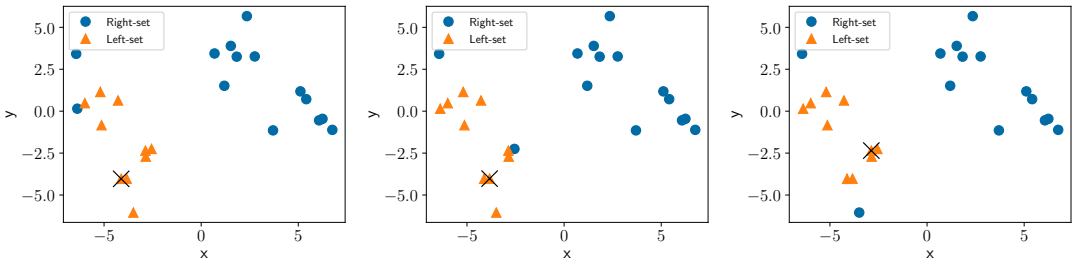
**Figure 5.4:** Depiction of the soft predictions that correspond to the nodes in Figure 5.3. We plot the SOE-embedding that we have calculated on the triplet data as a visualisation aid. Each cut corresponds to a node and thus to a cluster of a group of clusters. The color of a point corresponds to the probability that the point belongs to the given cluster(s), the darker, the more probable. In a), we see the coarse clustering that corresponds to the nodes directly below the root. On the left, we see the barn cluster, on the right the kitchen cluster. In b), we see the clusters that node 1F (the kitchen cluster) is made of. On the left, we have most of the kitchen items, on the right, we have the cluster of ktichen forks that look a bit more like pitchforks together with the clean-looking pitch forks (see Figure 5.2 and the corresponding discussion).

Next, to show that the last evaluation wasn't just a particular edge case, we repeat some of the basic evaluations for another observer. We select odd triplets and the thematic image set for person 3 and expect to see a behaviour similar to the one for person 3. In Figure 5.6 we plot the embedding from Schönmann again together with the tangles clustering. This time, we see three clusters (hay-straw, pitchforks and dishwashers-forks). These clusters coincide nicely with our classes, aside from one fork being clustered together with the pitchforks. We note that this is not the fork from Figure 5.2a that was clustered together with the pitchforks, so this might be a missclassification. If we look at the hierarchy (not plotted for space reasons) this time, we see that the pitchforks first get split off, and then the straw-hay from the kitchen dishwashers-forks cluster. This could again lead to interesting conclusions, which should probably be discussed by someone with more expert knowledge in the field, possibly using other evaluation data.




**(a)** Characterising cuts coarse cluster (kitchen-barn)

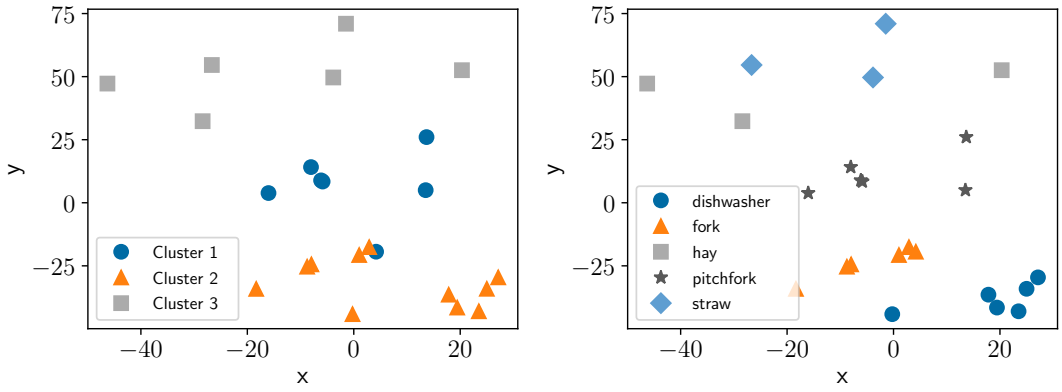
**(b)** Images of data points point inducing characterising cuts for coarse cluster




**(c)** Characterising cuts fine cluster (kitchen items - special forks)

**(d)** Images of data points inducing characterising cuts for fine cluster

**Figure 5.5:** Depiction of the characterising cuts of all splitting nodes on the thematic image data set. We draw only the orientation that corresponds to the left subtree, the orientation for the right one would just be all cuts reversed. This means, if a datapoint is often contained in the cuts that we depicted above, it is placed in the left subtree with high probability. In a), we plotted the characterising cuts for the root node (coarse split) and in c) have plotted the characterising cuts for the fine split (node 0F). As these cuts come from our original set of majority cuts, we have marked the data point that induced the particular cut with a black X. Below the characterising cuts in b) and d), we have plotted the images of the points that induced them.




**(a)** Original classes **(b)** Tangles clusters

**Figure 5.6:** Analog to Figure 5.1: on the left we plot the prediction of majority tangles with agreement 3 and radius 1 over a 2-dimensional SOE-embedding. On the right, we plot the original classes.

## 5.3 Discussion

In this chapter, we have shown that tangles can be used as an additional tool in evaluating triplet experiments, providing valuable insights. In particular, the more flexible clustering can show new dependencies between data points, as we made out a cluster of forks being perceived differently by observer 2. The hierarchy provided by tangles can help put these new dependencies into a better perspective, allowing us to gather that the pitch forks in the special fork cluster were perceived to belong more in a kitchen setting than in a barn setting. The explanations by tangles were used to make out the more defining items of a cluster (straw/hay for a barn, dishwashers for a kitchen). To our knowledge, hierarchical and explainable methods have not been used in psychophysics to this date, allowing tangles to eventually fill a gap.

However, we think that not all of the potential of tangles could be showcased here. As shown in our simulations, landmark tangles performs much better than majority tangles on all data sets. Also, landmark tangles can provide explanations that are more intuitive than those from majority tangles, as the cuts are inherently more explainble. A very interesting addition to our research would be applying tangles to our *ideal* real-world data set. This means that we have objects that exhibit a cluster structure, and that we have triplets sampled in a landmark format. To our knowledge, such a data set does not exist yet.

# Chapter 6

# Conclusion

In this work, we demonstrated a suitable extension for the tangles algorithm to apply it to clustering and hierarchy reconstruction. We validated its properties on simulated and experimental data under different external circumstances.

As we compared different algorithms in our experiments, we can also give recommendations as when to use which clustering algorithm. Especially, we can make out some conditions where tangles could provide a substantial benefit over others.

First off, if the goal is explainability, tangles can be a great choice. We could not find any other clustering algorithm that works with triplet data directly and produces explainable clusterings. An alternative could be using an explainable algorithm such as *explainable k-Means* (Moshkovitz et al., 2020) on the embedding created by an ordinal embedding directly, but this was out of the scope of this work. The importance of explainable algorithms is likely to increase, especially with a *right to explanation* shifting more into the focus of policy makers and legal scholars (Selbst and Powles, 2017), thus tangles could provide a substantial benefit over other algorithms for clustering triplets.

For standard clustering, we observed a very good overall performance of landmark tangles, provided the triplets were sampled in a landmark-approach. If data is already present in this format, we can recommend tangles as a clustering method, as it often shows better performance than the state-of-the-art algorithm SOE. This is especially true in the low-triplet regime. If a new experiment is to be designed, the experimentator might think about whether it is feasible to sample data in a landmark format to utilize tangles, especially if its other properties (explainability and hierarchy reconstruction) are desired. If the data is expected to be very noisy however, SOE might still outperform tangles.

If the data is not in landmark format, we only recommend using tangles (majority tangles in this case), if explainability and/or reconstructing hierarchies are desired, as SOE outperforms majority tangles in all our simulated

experiments.

In the case of hierarchy reconstruction, we can give a clear recommendation of tangles. We have seen that landmark tangles performs best out of our evaluated algorithms, and majority tangles comes directly afterwards at around the performance of soft ordinal embedding combined with average linkage. It can be problematic that tangles cannot construct a true dendrogram, which is what the hierarchical clustering literature focusses on (such as in Ghoshdastidar et al. (2019)). For practical applications however, the output from tangles can be good enough.

We also see that despite the setting not being optimal (no landmark triplets), we still receive good practical results with majority tangles in the setting of psychophyics, confirming and expanding on the insights that the original author had gained through an ordinal embedding. We envision that tangles could also be used as an additional tool supplementing an evaluation done via an ordinal embedding.

Through our work, we also uncovered some problems and potential new research directions for clustering triplet data with tangles. The tangles framework by Klepper et al. (2020) is very flexible and has multiple areas where it can be expanded or modified on. As we demonstrated, changing how to preprocess triplets to cuts is an effective method. Further work could explore other ways of doing this preprocessing step, which is still lacking for non-landmark triplets. A part of the tangles algorithm that we have not touched is the cost function. One could imagine different cost functions than the one we used, for example calculating some type of cost on the cuts using the triplet information. Additionally, one could think about modifying parts of the tangles framework themselves, which could maybe help with the problem of hierarchy noise we encountered in Subsection 4.3.4.

A big missing piece is still the performance of landmark tangles on real data. So far, we could not find any data sets that exhibit a useable cluster structure, and consisted of landmark triplets. This data could be easily generated in a controlled environment (for example using the approach from Schönmann (2021)), evaluated and compared to ordinal embeddings.

# Bibliography


S Agarwal, J Wills, L Cayton, G Lanckriet, D Kriegman, and S Belongie. Generalized Non-metric Multidimensional Scaling. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.

J Anderton and J Aslam. Scaling Up Ordinal Embedding: A Landmark Approach. *International Conference on Machine Learning (ICML)*, 2019.

S Balakrishnan, M Xu, A Krishnamurthy, and A Singh. Noise Thresholds for Spectral Clustering. *Neural Information Processing Systems (NeurIPS)*, 2011.

Ç Demiralp, M Bernstein, and J Heer. *Learning Perceptual Kernels for Visualization Design*, volume 20. 2014.

R Diestel. Tangles in the social sciences. *arXiv:1907.07341 [physics]*, 2019.

R Diestel and G Whittle. Tangles and the Mona Lisa. *arXiv preprint arXiv:1603.06652*, 2017.

E Fluck. Tangles and single linkage hierarchical clustering. In P Rossmanith, P Heggernes, and J.-P Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, 2019.

N Ghosh, Y Chen, and Y Yue. Landmark Ordinal Embedding. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

D Ghoshdastidar, M Perrot, and U von Luxburg. Foundations of Comparison-Based Hierarchical Clustering. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

S Haghiri, P Rubisch, R Geirhos, F Wichmann, and U von Luxburg. Comparison-Based Framework for Psychophysics: Lab versus Crowdsourcing. *arXiv preprint arXiv:1905.07234*, 2019.

S Haghiri, F. A Wichmann, and U von Luxburg. Estimation of perceptual scales using ordinal embedding. *Journal of Vision*, 20(9):14–14, 2020.

L Jain, K. G Jamieson, and R. D Nowak. Finite Sample Prediction and Recovery Bounds for Ordinal Embedding. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

M Kleindessner and U von Luxburg. Kernel functions based on triplet comparisons. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

M Kleindessner and U von Luxburg. Lens Depth Function and k-Relative Neighborhood Graph: Versatile Tools for Ordinal Data Analysis. *Journal of Machine Learning Research*, 18(58):1–52, 2017.

S Klepper, C Elbracht, D Fioravanti, J Kneip, L Rendsburg, M Teegen, and U von Luxburg. Clustering with Tangles: Algorithmic Framework and Theoretical Guarantees. *arXiv preprint arXiv:2006.14444*, 2020.

L. van der Maaten and K. Weinberger. Stochastic triplet embedding. *IEEE International Workshop on Machine Learning for Signal Processing*, 2012.

Z Liu and R Modarres. Lens data depth and median. *Journal of Nonparametric Statistics*, 23(4):1063–1074, 2011.

S Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

M Moshkovitz, S Dasgupta, C Rashtchian, and N Frost. Explainable k-Means and k-Medians Clustering. *International Conference on Machine Learning (ICML)*, 2020.

B. D Roads and B Love. Enriching ImageNet with Human Similarity Judgments and Psychological Embeddings. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

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

I Schönmann. *Similarity Judgements of Natural Images: Instructions Affect Observers' Decision Criteria and Consistency.* Bachelor thesis, Universität Tübingen, 2021.

A. D Selbst and J Powles. Meaningful information and the right to explanation. *International Data Privacy Law*, 7(4):233–242, 2017.

R. N Shepard. The analysis of proximities: Multidimensional scaling with an unknown distance function. II. *Psychometrika*, 27(3):219–246, 1962.

O Tamuz, C Liu, S Belongie, O Shamir, and A. T Kalai. Adaptively learning the crowd kernel. *International Conference on Machine Learning (ICML)*, 2011.

Y Terada and U Luxburg. Local ordinal embedding. *International Conference on Machine Learning (ICML)*, 2014.

A Ukkonen. Crowdsourced Correlation Clustering with Relative Distance Comparisons. *International Conference on Data Mining (ICDM)*, 2017.

L van der Maaten and G Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

L. C Vankadara, S Haghiri, M Lohaus, F. U Wahab, and U von Luxburg. Insights into Ordinal Embedding Algorithms: A Systematic Evaluation. *arXiv preprint arXiv:1912.01666*, 2021.

# Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum Unterschrift