Eberhard Karls University of Tübingen
Department of Mathematics and Natural Sciences
Wilhelm Schickard Institute for Computer Science

Master Thesis Computer Science

Clustering Triplets with Tangles

Alexander Conzelmann

18.10.2022

Reviewers

Prof. Dr. Ulrike von Luxburg
Theory of Machine Learning
Wilhelm Schickard Institute for Computer Science
University of Tübingen

Prof. Felix Wichmann, DPhil
Neural Information Processing
Wilhelm Schickard Institute for Computer Science
University of Tübingen

Conzelmann, Alexander
Clustering Triplets with Tangles
Master Thesis Computer Science
Eberhard Karls University of Tübingen
Thesis period: 04/2022–10/2022

Abstract

We investigate a new algorithm for clustering triplets (comparison-based data without absolute distance information). This new approach is based on the tangles framework, a tool previously used to find dense structures in graphs, which we extend to work on triplets. Current work on clustering triplets focusses on embedding the triplets 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 fewer distortions on our data and achieving a higher clustering accuracy. In addition to being competitive in performance, our approach provides both explainability as well as a cluster hierarchy on top with no added cost. We evaluate the tangles algorithm both on synthetic data under a diverse range of settings as well as experimental data from the realm of psychophysics.

Zusammenfassung

Wir untersuchen einen neuen Algorithmus, um Triplets zu clustern. Triplets sind Daten, die Vergleichen zwischen Datenpunkten basieren, jedoch keine absoluten Distanzinformationen beinhalten. Unser neuer Ansatz basiert auf dem Tangles-Framework, welches in der Vergangenheit verwendet wurde, um dichte Strukturen in Graphen zu untersuchen. Wir erweitern in dieser Arbeit den Tangles-Algorithmus, um ihn auch auf Triplets anzuwenden. Die bisherige Forschung zur Clusterung von Triplet verwendet größtenteils ordinale Einbettungen. In diesen werden die Triplet zuerst in einen euklidischen Raum eingebettet, was die Daten verzerren kann. Danach wird ein klassischer Clustering-Algorithmus verwendet. Im Gegensatz dazu verwendet unsere Methode keine euklidische Zwischenrepräsentation. So können wir potentiell Verzerrungen in den Daten umgehen und ein präziseres Clustering generieren. Unser Ansatz zeigt nicht nur eine kompetitive Leistung, sondern bietet auch die Möglichkeit, die Ergebnisse zu erklären und hierarchisch aufzuschlüsseln. Wir evaluieren den Tangles-Algorithm auf synthetischen Daten in einer Reihe von verschiedenen Szenarien und außerdem auf experimentellen Daten aus dem Forschungsgebiet der Psychophysik.

Acknowledgements

I would like to thank Prof. Ulrike von Luxburg for welcoming me so warmly into her group and for providing me room to grow as a researcher.
I would also like to extend my deepest gratitude to Solveig Klepper for countless inspiring discussions, for always providing valuable insights and for her great attention to detail.
Thanks also to David-Elias Künstle for his help whenever I needed to know something about psychophysics.
Lastly, I want to thank all my friends who were always there for me, and I especially thank Levin Güver for helping me with all the little details of research work and for always providing an interested audience when I needed to explain wild ideas to someone.

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 humans 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, together with a suitable question. For example, we might draw the colors violet, red and yellow, and ask the observer *is violet more similar to red or yellow?* These questions are comparatively easy to answer for humans. What remains is the question of how to evaluate the obtained triplets.

A small research community has formed around the task of dealing with triplets. Most of this community focuses 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 respect as many of the original triplets 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 can for example be the case if the triplets are created using a distance function that does not obey the triangle inequality. Another possible complication can be contradicting triplets, which often occur

when gathering data from human observers.

However, a big advantage of ordinal embeddings is exactly 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 triplets 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 (Lloyd, 1982), on this representation. An example of this approach can be seen in Kleindessner and von Luxburg (2017). But, 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 paper, 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 is suitable for hierarchical clustering. Central objects in this framework are cuts, which are ways of dividing a set into two distinct, non-overlapping subsets. To cluster data using the tangles framework, one first needs to obtain a set of cuts on the data. These cuts are required to 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 process triplets to cuts suitable for the tangles algorithm. Using these methods, we demonstrate that the tangles algorithm can be successfully applied to (hierarchically) cluster triplets without creating an intermediate ordinal embedding. We evaluate our approach by simulated and experimental data and show that it is competitive in performance to approaches based on ordinal embeddings, 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 choose a real-world triplet data set from the realm of psychophysics and evaluate our algorithms on this data.

Chapter 2

Theoretical Background

In this chapter, we introduce the theoretical concepts used in this work. In particular, we give an in-depth explanation of tangles, triplet data, and state-of-the-art methods of evaluating triplet data.

2.1 Tangles

Tangles, introduced originally by Robertson and Seymour (1991), have been used in mathematics to study highly cohesive structures in graphs. 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 use the definitions introduced therein.

In this section, we 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 $V = \{v_1, v_2, \dots v_n\}$ into

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

$$A \cap B = \emptyset \text{ and } A \cup B = V. \tag{1}$$

We usually write a cut as $P = \left(A, \overline{A}\right)$, with $A \subset V$ and $\overline{A}$ being the complement of A with respect to V. As a side note, $\left(A, \overline{A}\right)$ and $\left(\overline{A}, A\right)$ are equivalent cuts, the order of the two elements only matters when it comes to orientations (see below).

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. Concretely, an informative cut should not separate groups of data that are tightly coupled. On a graph, an informative cut $P = \left(A, \overline{A}\right)$ 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* $c : \mathcal{C}(V) \to \mathbb{R}$, with $\mathcal{C}(V)$ denoting the set of all possible cuts on 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 between the nodes of the two sets $A, \overline{A}$.

Orientation. An *orientation* o of a cut $P = \left(A, \overline{A}\right)$ is a choice of either A or $\overline{A}$. We call a cut *left* oriented if we pick A and *right* oriented if we pick $\overline{A}$. An *orientation* of a set of cuts $\mathcal{B} = \left\{\left(A_1, \overline{A_1}\right), \ldots, \left(A_n, \overline{A_n}\right)\right\}$ 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 or $\overline{A_i}$.

Consistency condition. Assume that for a set of elements V we have a set of cuts $\mathcal{B} = \left\{\left(A_1, \overline{A_1}\right), \ldots, \left(A_n, \overline{A_n}\right)\right\}$ 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 usable form by the tangles framework. For this, we find in $\mathcal{B}$ the set of *tangles*. These correspond to specific ways of orienting the cuts such that they point to cohesive structures in the data. A tangle is an orientation for which:

$$\forall A, B, C \in O : |A \cap B \cap C| \geq a, \tag{2}$$

for some fixed parameter $a \in \mathbb{N}$, which we refer to as *agreement* parameter. Equation 2 is also called the *consistency condition*.

Order. Using this 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 cut halves 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_ψ, using a threshold $\psi \in \mathbb{R}$ such that

$$P_\psi = \{c(P) \leq \psi\}. \tag{3}$$

A tangle on P_ψ is said to have *order* ψ.

To illustrate some of the concepts better, we include a schematic drawing of a tangle with an agreement of 3 on a simple data set composed of two clusters in Figure 2.1. Here, we might already gain some intuition on why tangles can find dense structures in data. The tangle that is depicted in the figure orients all cuts left (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 the right, else the consistency criterion is violated, as then the intersection of the orientations of the cuts contains at most one point. All in all, there is exactly one tangle for each cluster.

2.1.2 Processing tangles to a clustering

In Figure 2.1, each of the tangles we found 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 for how clusters can be identified through tangles. We assume that we have chosen an appropriate cost function c and an agreement parameter a.

We are given a tuple of cuts $\mathcal{B} = \left(\left(A_1, \overline{A_1}\right), \ldots, \left(A_n, \overline{A_n}\right)\right)$ which are sorted in ascending order according to the cost function c. Next, we build a tree structure on 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 iteratively build the tree: 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 concerning a, we add the orientation $n_j \cup A_k$ as a left child. We then try to add the cut in the right orientation as well and append $n_j \cup \overline{A_k}$ as the 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 threshold $\leq \psi_k$ 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

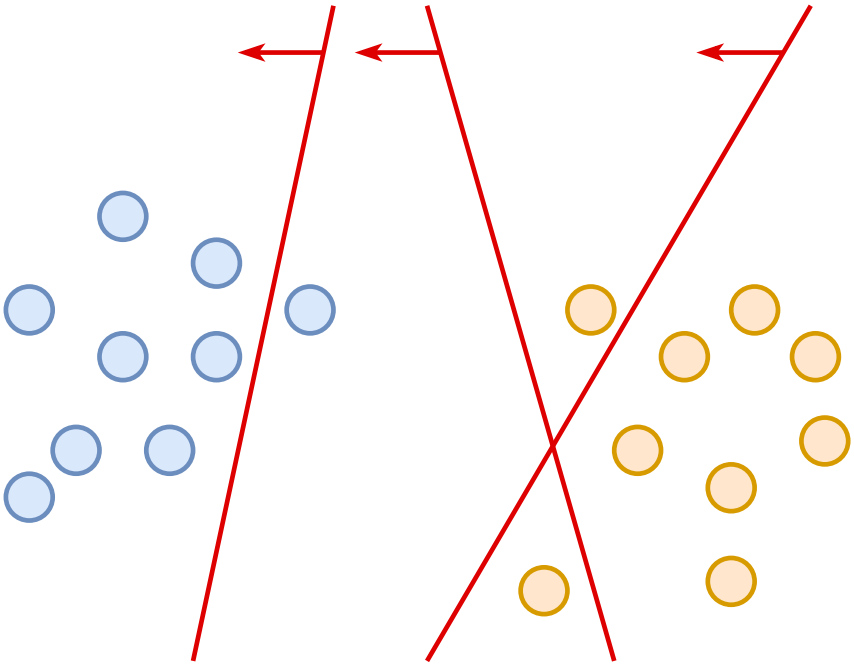


Figure 2.1: A simple tangle for a reasonably sized agreement ($a = 3$). The data set consists of two clusters, one left (blue), and one right (orange). We assume that we have obtained three cuts on the data set, represented by red lines. We draw a possible orientation on the cuts, indicated by the red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

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 the set $\{P_1, P_2, P_3\}$. As another example, there exist 3 tangles on $\{P_1, P_2\}$.

We now discuss 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 tangle 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. Concerning 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 that the tangle points to further. In the end, each leaf node will correspond to exactly one cluster and vice versa.

Splitting nodes represent meaningful paths in our tree. If a node only has

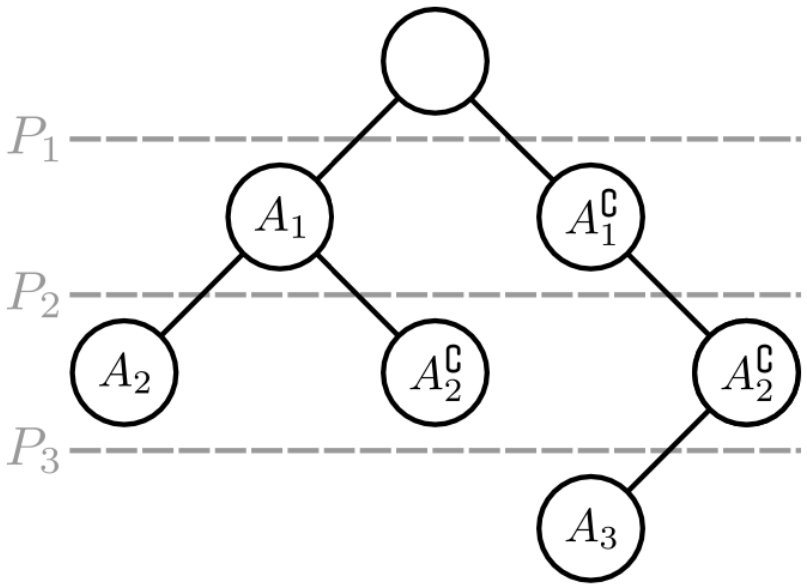


Figure 2.2: A possible tangle search tree for a set of cuts $\mathcal{B} = \{(A_1, \overline{A_1}), (A_2, \overline{A_2}), (A_3, \overline{A_3})\}$. The nodes on a level each correspond to a tangle of the order given by the cut P_i that is written on the dashed line to the left. Figure taken with permission from Klepper et al. (2020).

one child, the cut that was added last could only be added in one orientation. This cut might make it harder to add more cuts further down the tree, but it does not present a meaningful division of our cluster. 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.

For the splitting nodes, we want to know which cuts determine what cluster a point belongs to. An initial idea would be to only take 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. Thus, one determines the set of *characterizing cuts*. A cut belongs to this set if it is both oriented the same way inside each subtree, and oriented in a different way between the two subtrees. Thus, all characterizing cuts make a meaningful decision between the left and right subtree and are coherent in their decision inside each 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 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 tangle search tree. *Soft* means that every data point is assigned a

probability of belonging to each cluster. *Hierarchical* means that we have a hierarchy of the resulting clusters, which arises naturally from the form of the tangle search tree. For a soft clustering, we determine the probability of a point x belonging 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 characterizing 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 characterizing cuts at the splitting node, we receive the probability p_L that x belongs to the left cluster. As the characterizing 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 probabilities 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 triplets

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 the 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:

$$d(a, b) < d(c, d). \tag{4}$$

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

$$d(a, b) < d(a, c). \tag{5}$$

A triplet (a,b,c) can also be expressed by the quadruplet (a,b,a,c), making quadruplets strictly more general. However, for the rest of this work, we use triplets to describe ordinal constraints, as they are a bit simpler to work with and still expressive enough for our purposes.

Datasets of triplets are almost always obtained by asking human participants. For example, we might collect triplet data on images by presenting human participants with three images a, b, c, and asking 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 be transformed back to triplet format for further processing. 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 feature-based data (k-Means, support vector machines, neural networks, et cetera). Triplets are an unusual data form, which is why one of the most common evaluation methods is to first use an ordinal embedding algorithm on the triplets to transform the data points into euclidean space, before processing it further. Therefore, we 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 do not 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 triplets consists of finding a so-called *ordinal embedding* of the data. If we have a set of triplets $T = \{t_1, t_2, \dots t_n\}$, 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 the metric. Formally, we want to minimize (Vankadara et al., 2021)

$$\min_{y_1,\dots y_n \in \mathbb{R}^m} \sum_{t=(i,j,k)\in T} \mathbb{1}_{\|y_i - y_j\|_2 < \|y_i - y_k\|_2}. \tag{6}$$

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 δ 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 consider a set of quadruplets Q on a 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, δ 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 λ 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, ξ_{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 α degrees of freedom:

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

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 satisfied: 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 lens of two faraway points. Using a specialized 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 neighborhood 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 neighborhood 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.

- Kleindessner and von Luxburg (2017) uses the triplets directly to estimate a kernel function between the points in the dataset. They present two different kernel functions. In both approaches, they start by determining a similarity between all pairs of points x and y out of the dataset X. For the first kernel function k_1, they rank all other points $x \in X$ by their closeness to x and repeat the same for y. Then, they take the Kendall tau correlation coefficient (which is a kernel function on the set of total rankings) between the two rankings and use this to generate a feature map that serves as the value of the kernel function $k_1(x, y)$.

  For the second kernel function k_2, they determine how similar two points x and y are by counting the number of triplets that they agree on. They do this by determining all pairs of points x_i, x_j for which both (x_i, x_a, x_j) and (x_i, x_b, x_j) hold. From the number of these pairs, they subtract the number of all pairs for which either (x_i, x_j, x_a) and (x_i, x_b, x_j) or (x_i, x_a, x_j) and (x_i, x_j, x_b). This similarity is used in a feature map from which they construct $k_2(x, y)$.

  The kernel functions can then be used in any kernel machine, such as kernel SVM for classification or regression.

- Ghoshdastidar et al. (2019) presents methods to hierarchically cluster data from a set of quadruplets Q. They argue that single-linkage and complete-linkage can be naturally implemented for a quadruplet setting, as they only require us to know which objects are the closest together or farthest away from each other, without requiring absolute distance values. As single-linkage and complete-linkage have weak statistical guaran-

tees, they present two methods to implement average linkage for quadruplet data. For the first one, they use a kernel similar to Kleindessner and von Luxburg (2017) to estimate similarities between the objects using the quadruplets Q and then use the standard linkage procedure (iteratively merge the two most similar clusters). For the second one, they estimate whether two clusters G_1, G_2 are more similar to each other than two clusters G_3, G_4 directly using the quadruplets. This is done by iterating over all indices i, j, k, l for which $x_i \in G_1, x_j \in G_2, x_k \in G_3, X_l \in G_4$, adding up the number of quadruplets $(i, j, k, l) \in Q$ and subtracting the number of quadruplets $(k, l, i, j) \in Q$. This similarity between clusters can then again be used in a standard linkage procedure.

Chapter 3

Clustering Triplet Data with Tangles

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 chapter, 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 use the one by Haghiri et al. (2019). Assume we have a set of objects D, as well as a set of triplets T, which 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

$$\forall l_i, l_j \in L \; \forall x \in D : (x, l_i, l_j) \in T \vee (x, l_j, l_i) \in T. \tag{7}$$

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 $(A, \overline{A})$ 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

$$\text{Land}_{ij} = \{x \in D \mid (x, l_i, l_j) \in T\}, \tag{8}$$

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:

$$\forall (x, y) \in L' \, \forall x \in D : (x, y, z) \in T \vee (x, z, y) \in T.. \tag{9}$$

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 some cluster information: the closer x is to l_i according to d, the more likely it is that 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 i, j, as illustrated in Figure 3.1.

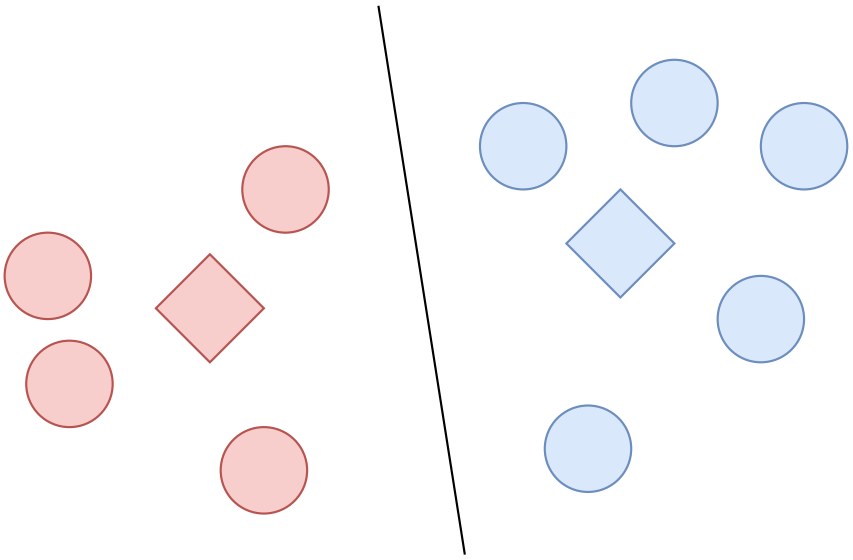


Figure 3.1: A landmark cut on a euclidean data set. The two diamonds are the landmarks l_1 (red) and l_2 (blue). All red points (left of the line) are contained in $\text{Land}_{1,2}$, all blue points in $\overline{\text{Land}_{1,2}}$.

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 that is both sampled according to a landmark approach and exhibits a cluster structure. We thus rely on simulations for testing landmark cuts. These simulations can be found in Chapter 4.

To make landmark cuts work with a set of triplets that are sampled uniformly at random, one can impute the missing triplets with standard methods. An example would be random imputation. There, one would randomly either add (a, b, c) or (a, c, b) to the set of triplets, if none of them were present. We will explore this more in-depth in Chapter 4.

3.2 Majority cuts

As explained in Section 3.1, sampling triplets 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 more likely belong to the same cluster. Based on this, a cut $P = \left(A, \overline{A}\right)$ is more informative 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 in a metric space would be 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 similarity (they all have a maximum distance of $2r$ 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

$$L_x = \{t \in T \mid t = (x, b, c), b, c \in D\}. \tag{10}$$

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

$$M_x = \{t \in T \mid t = (a, x, c), a, c \in D\} \tag{11}$$
$$R_x = \{t \in T \mid t = (a, b, x), a, b \in D\}, \tag{12}$$

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

$$\mathrm{Maj}_a := \{x \in D \mid |L_a \cap M_x| < |L_a \cap R_x|\}, \tag{13}$$

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

$$\forall a, b, c \in D : (a, b, c) \in T \vee (a, c, b) \in T. \tag{14}$$

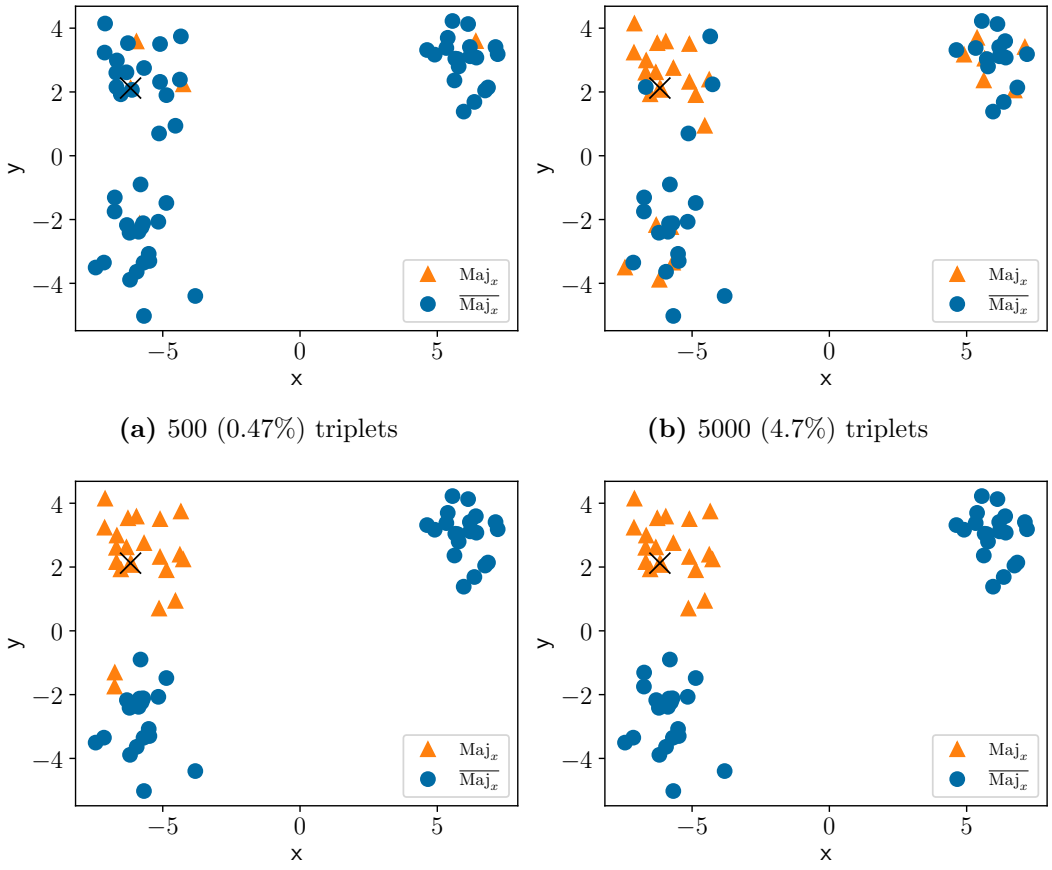
then Maj_a is a ball around a whose radius is the median distance of a to all other points $x \in D$, as only those points will appear more often closer to a in the triplets than they appear farther away. When T does not contain all possible triplets, we introduce noise on our cuts and Maj_a then contains 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

$$\text{Maj}_a(r) := \{x \in D \mid |L_a \cap M_x| < r \cdot |L_a \cap R_x|\}. \tag{15}$$

and call r the *radius* of the cut. $\text{Maj}_a(r)$ 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 ball around a becomes corrupted by noise.

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.



(a) 500 (0.47%) triplets

(b) 5000 (4.7%) triplets

(c) 20000 (18.8%) triplets

(d) 106200 (100%) triplets

Figure 3.2: Majority cuts more closely resemble a ball around their anchor point as the number of sampled triplets increases. We plot Maj_a with radius $r = 0.5$, generated according to the procedure in Section 3.2. The black X marks the anchor point a. The set Maj_a consists of the orange triangles, which are the data points that are twice as often closer to a than they are not (according to the drawn triplets). The blue circles are the points not in Maj_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. For this chapter, 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. In both cases, we generate triplets from the data points. The gaussian setup serves as a first baseline: it is a standard task in clustering and a lot of real-life data is approximately gaussian. It is also sufficiently simple and allows us to focus our attention on how the triplets are generated (which can be corrupted by noise, missing entirely, et cetera). The noisy hierarchical block matrix was introduced by Balakrishnan et al. (2011). We use it to illustrate another property of tangles: in addition to pure clustering, we also produce a hierarchical tree, which naturally induces hierarchical clustering.

4.1 Terms and methods used

Cut generation and tangles. We evaluate two different approaches to the tangles algorithm, which are based on the two different methods of generating cuts that we detailed in Chapter 3:

- *Landmark tangles* (L-Tangles), where we process the set of triplets to a set of landmark cuts and then apply tangles on the resulting cuts. This approach is straightforward if the triplets are sampled in a landmark fashion. If we have a set of data points X, and a set of triplets T, the triplets are sampled in a landmark fashion, if we have a set of of tuples $\{(x_i, x_j) \mid x_i, x_j \in X\}$ for which:

$$\forall a \in X : (x_i, x_j, a) \vee (x_i, a, x_j).$$

If the triplets are not sampled in a landmark fashion, we can opt to impute the missing triplets. We use three different imputation methods: random, 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.

- *Majority tangles* (M-Tangles), where we process the set of triplets to majority cuts and apply tangles on these. This approach is suitable for all sets of triplets, regardless of how they were sampled.

Cost function. We will use a very flexible cost function in our simulations, called the *mean manhattan cost function*. It generalizes well to arbitrary cuts and does not rely on additional information (such as edge weights in a graph clustering setting). Assume we have a set of points X and a set of cuts $\mathcal{B} = \{(A_1, \overline{A_1}), (A_2, \overline{A_2}), \dots (A_n, \overline{A_n})\}$ on X. We first define a similarity between points using the set of cuts:

$$s(u, v) = \sum_{i=0}^{n} \mathbb{1}_{(u \in A_i \wedge v \in A_i) \vee (u \in \overline{A_i} \wedge v \in \overline{A_i})}, \tag{16}$$

where $\mathbb{1}$ is the indicator function. We then define the mean-manhattan cost function as:

$$c((A, \overline{A})) = \frac{1}{|A| \cdot |\overline{A}|} \sum_{u \in A, v \in \overline{A}} s(u, v). \tag{17}$$

Triplet sampling. 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 c than to b) holds. Then, we use two different approaches to 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 repeatedly draw two a, b with $a \neq b$ uniformly at random, and then sample all triplets that have the form (x, a, b) or (x, b, a). For landmark tangles, the second kind of triplets is preferred, 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). Second, we can reduce the total amount of triplets sampled, for which we use the term *density*. If we sample triplets with a density of d, that means we sample only a fraction d of all triplets.

Evaluation metrics. To evaluate the performance of the tangles algorithms, we also need metrics on the performance. 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), and 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. To obtain the AARI for two hierarchies, we calculate the ARI over all levels of them and then average over all the obtained scores.

Baselines. We use multiple baselines against which we compare landmark and majority tangles. An idea is to use a combination of an ordinal embedding together with a classical clustering algorithm, an approach that has also been used in Kleindessner and von Luxburg (2017). 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. This was also the case for us when comparing different baseline algorithms. We also include t-Stochastic Triplet Embedding (t-STE, L. van der Maaten and K. Weinberger, 2012) as another ordinal embedding algorithm. For the clustering algorithm, we use k-Means (Lloyd, 1982), as this is one of the most established clustering algorithms and has a good performance across a wide variety of data sets.

As another baseline, we included ComparisonHC (Ghoshdastidar et al., 2019), which we use with the quadruplets kernel average linkage method (4K-AL) that the authors introduced. Like tangles, this algorithm clusters the triplets directly, without constructing an intermediate embedding. As an output, ComparisonHC produces a dendrogram, which is normally used to identify a cluster hierarchy. However, by cutting the dendrogram at a certain level such that it produces the desired amounts of clusters, one can also obtain a non-hierarchical clustering.

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 gaussian distributions with means $\mu_1, \mu_2, \dots \mu_k$ and covariances $\Sigma_1, \Sigma_2, \dots \Sigma_k$. 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

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

with I as the identity matrix. A sample draw of the data set can be seen in Figure 4.1. The data might be trivial to cluster in a classical setting, as all the clusters are cleanly separated from one another by a rather large distance. This is not concerning for us: how large the distances between the points are does not matter, as the triplets we generate from the points contain only qualitative information. Additionally, the task of clustering the data gets sufficiently hard once we corrupt the triplets by noise or generate only a few triplets from the data.

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, following Klepper et al. (2020). For majority tangles, we use a radius of $r = \frac{1}{3}$, such that the cuts roughly have the diameter of the clusters. In both cases, we use the mean manhattan cost function (see Equation 17 in Subsection 2.1.1).

All results are the average of 50 runs, with the following procedure: We randomly sample points from the mixture of gaussians, then we sample a random set of triplets (applying noise if necessary), and lastly, we sequentially evaluate each algorithm on this set of triplets.

4.2.2 Lowering density

Setup. We draw an increasing amount of triplets (without noise) in a landmark fashion from the dataset. The fraction of all triplets drawn is denoted by the density d. We repeat the experiment for a small (60 points) and a large (600 points) dataset. Tangles are applied with an agreement of $a = 7$ for the small data set and $a = 70$ for the large dataset. With the larger dataset, we could not test ComparisonHC, as the implementation we have obtained requires constructing a n^4 matrix during the training step.

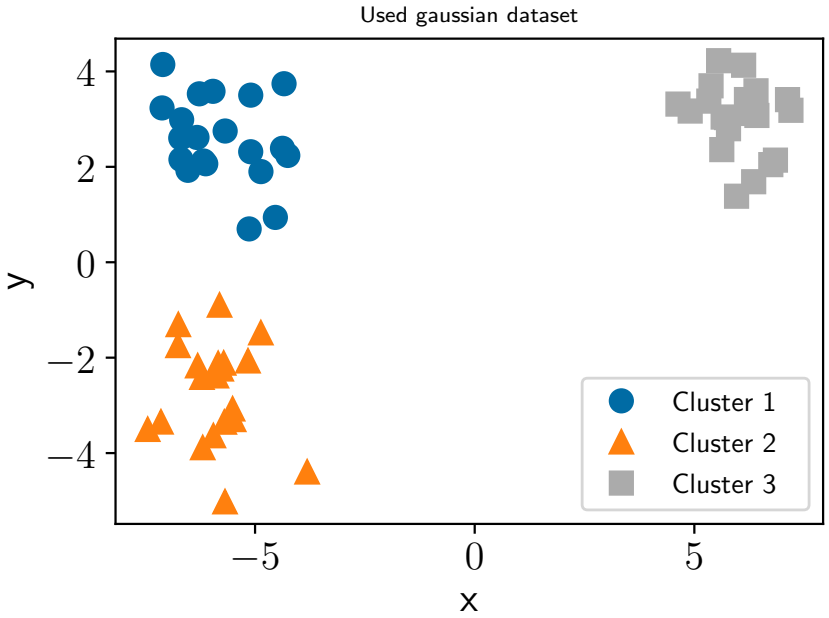
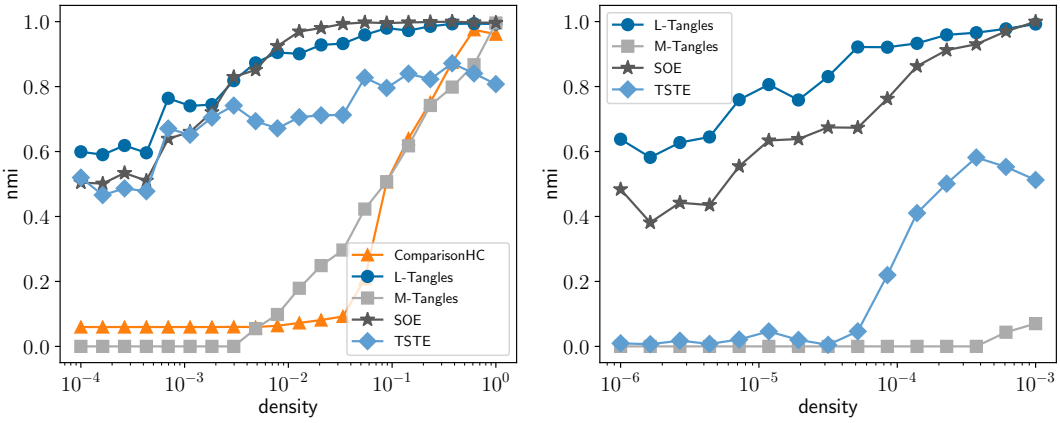


Figure 4.1: Example draw of the gaussian mixture used for our experiments.

Results. The results can be seen in Figure 4.2. 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 $n = 200$. 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.

Discussion. With this experiment, we want to observe how our methods behave under different numbers of triplets present. As the number of triplets 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.

Usually, the larger the dataset, the smaller the percentage of all triplets we use. 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). As can be seen in the results, tangles show the desired behavior of requiring a smaller percentage of total triplets with a higher number of data points.



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

Figure 4.2: L-Tangles shows competitive performance over various densities. 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

Setup. We draw all triplets (density 1.0) in a landmark fashion from the small (60 points) gaussian data set. We then corrupt the triplets with noise as follows: on a noise level of n, each triplet is flipped with probability n, meaning that (a, b, c) would be turned to (a, c, b). We evaluate the NMI of all clustering methods over differing noise levels.

Results. The results can be seen in Figure 4.3. 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. t-STE performs on a level between ComparisonHC and SOE, but interestingly struggles to achieve a perfect clustering (NMI of 1.0) of the data even for low noise.

Discussion. We observe how tangles behave under added noise. This is an important model: most applications of triplet data use triplets that are gener-

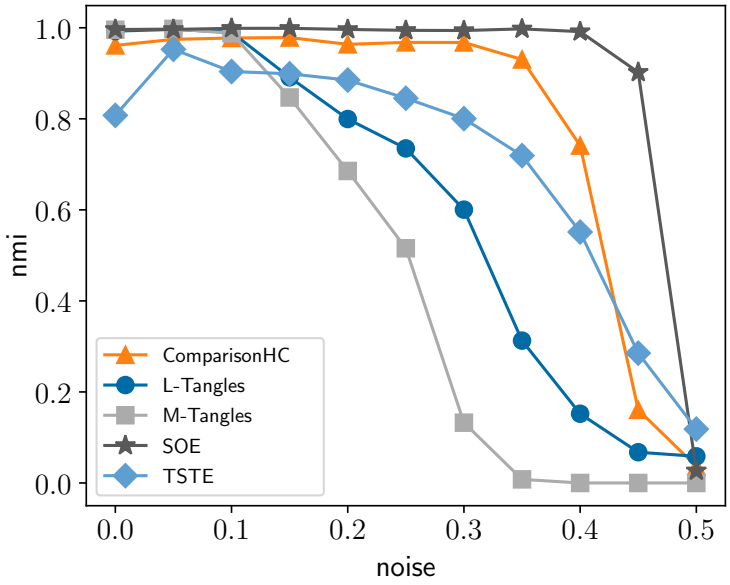


Figure 4.3: The performance of tangles falls off more quickly than other methods with increased noise, but still shows good performance for low-medium noise levels. We plot the NMI of our chosen clustering methods against the noise that we introduce on the triplets. We use 3 clusters and 20 data points 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)).

ated from real humans. They might disagree on which objects are closer and which are not, which can be modeled 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 rather than which objects you present to them.

In the results, we have seen that the performance of L-Tangles and M-Tangles falls off quicker than all other algorithms. Why this happens is not entirely clear, especially as tangles have shown good performance with cuts that were generated in a *quick-and-dirty* manner for graph clustering. We speculate that the noise in this triplet setting has a fundamentally different effect on the generated cuts. Let us compare the setting of graph clustering using tangles and triplet clustering using landmark tangles. When we generate cuts using a greedy algorithm such as the Kernighan-Lin algorithm (Kernighan, Brian Wilson and Lin, Shen, 1970), we probably end up with some less-informative cuts (that cut through densely connected regions), but also with some more-informative cuts. Using the cost function, the tangles algorithm sorts these cuts and produces a high-quality clustering by first aligning the more-informative cuts and then the less-informative cuts.

When we now generate cuts on triplets with low noise, the same thing hap-

pens: we receive some informative cuts and some less-informative cuts. Examples might be a landmark cut between two elements from different clusters (more informative) and a cut between two neighboring points (less informative). However, once we add noise to the triplets, we corrupt all of the cuts we generate. If we now generate cuts on these noisy triplets, we will end up with some medium-informative cuts at best (and some even less informative ones). One theory that would still have to be inspected more closely, is that tangles require at least a few very informative cuts and cannot correctly cluster data if we only have cuts of medium quality, which might be what we see in the results of this experiment.

4.2.4 Adding noise and lowering density

Setup. We combine the experiments done in Subsection 4.2.2 and Subsection 4.2.3 by drawing triplets in a landmark format for varying noise n and density d. We evaluate L-Tangles (applied with an agreement of $a = 7$) and SOE by calculating the NMI of their clusterings and plotting the values in separate heat maps.

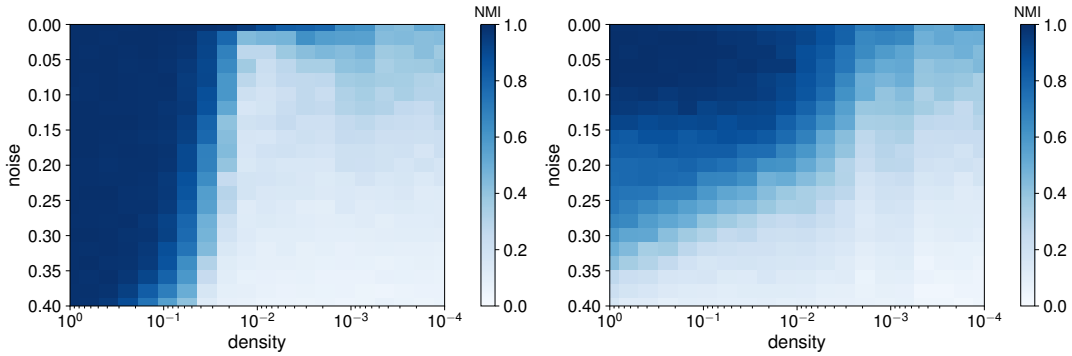
Results. The resulting heat maps can be seen in Figure 4.4; darker shades indicate better performance. There, we can see that L-Tangles outperforms SOE in quite a large region in the low-noise, low-density regime. On the contrary, SOE performs better in the high-density, high-noise regions, with about similar performance for the cases of low-noise high-density (perfect clustering) and high-noise low-density (random clustering).

Discussion. This result seems sensible when looking at the individual results in Subsection 4.2.2 and Subsection 4.2.3. There, we have seen that L-Tangles outperforms SOE for no noise over varying densities, and SOE outperforms L-Tangles for density 1.0 over various noise levels. The number of triplets and the noise on them are probably the two most important variables when dealing with triplet data. Thus, the fact that there is a sizeable region of densities and noise levels where L-Tangles outperforms SOE reinforces the utility of L-Tangles as a practical algorithm for clustering triplets.

4.2.5 Missing triplets and imputations

Setup. 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.

For this setup, we need to impute missing triplets in L-Tangles to use them. To do this, we generate landmark cuts as before, but we mark entries for which we don't have triplet information. These missing values are then imputed via



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

Figure 4.4: L-Tangles show a performance advantage over SOE in the low-density, low-noise regime. We plot a heatmap of the NMI of (a) SOE and (b) L-Tangles over noise (y-axis) and density (x-axis) of 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.

different methods. We have used *random*, *k-nearest neighbour* (k-NN) and *mean* imputation.

Results. We show the results for different imputation methods in Figure 4.5. There, we can see that 1-NN performs vastly better over all densities compared to the other imputation methods.

We compare the performance of tangles to other clustering algorithms in Figure 4.6. L-Tangles was imputed with 1-NN, as this was the best imputation method in terms of the achieved NMI score on the data set. 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

Discussion. Interestingly, a simple imputation method such as 1-NN shows the best imputation performance. This allows a straightforward choice of an imputation method that shows acceptable performance for high densities. However, in this experiment, L-Tangles and M-Tangles show a comparable performance for the first time, although L-Tangles is now on the level of M-Tangles and ComparisonHC and thus considerably worse than SOE. Sampling triplet in a landmark fashion and using L-Tangles seems to be overall preferable for performance.

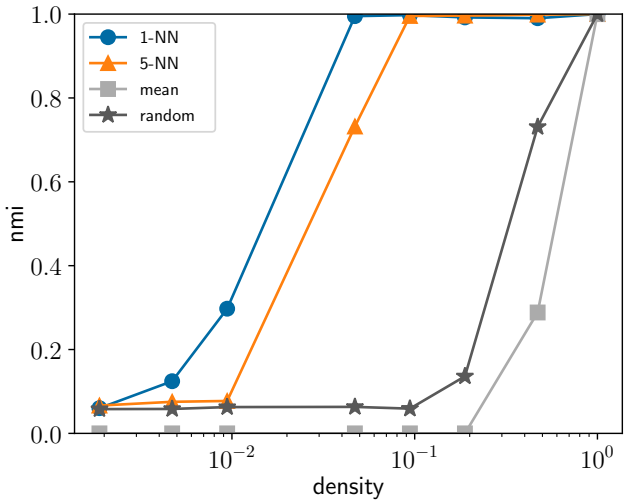


Figure 4.5: 1-NN performs well for imputing missing triplets for L-Tangles. We plot the performance of L-Tangles over the number of triplets available for different imputation methods over our small gaussian dataset with 3 clusters and $n = 20$. We use different imputation methods to fill in the missing values and then cluster the resulting cuts with L-Tangles. The *k-NN* methods use k-nearest neighbor 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.

An interesting area of further research would be devising more effective imputation schemes, for example using heuristics like the triplet kernel presented in Kleindessner and von Luxburg (2017).

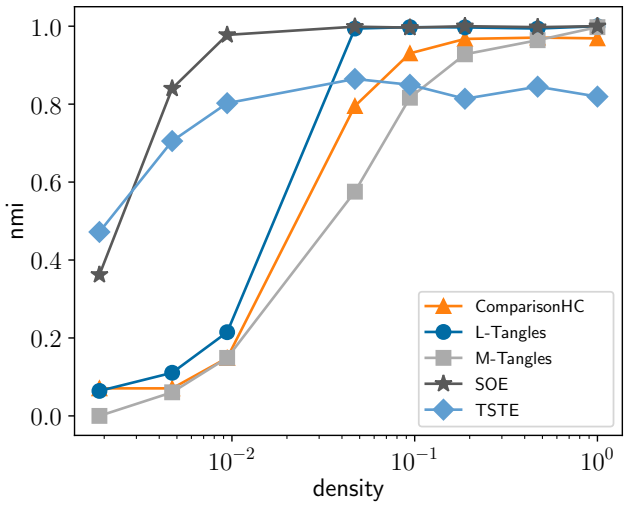


Figure 4.6: L-Tangles show an overall worse performance than when triplets are drawn in a landmark fashion. Analog to Figure 4.5, where we visualize the performance of clustering algorithms for triplets drawn uniformly at random. This time we plot the performance of other clustering algorithms on the small gaussian data set against L-Tangles. We impute the missing values in the L-Tangles algorithm with 1-NN.

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 x_i, x_j. In this matrix, the elements in the same cluster have the highest similarity μ_0. If two elements x_i, x_j are from different clusters, their similarity is calculated as

$$\mu_{i,j} = \mu_0 - (h - l(a_{i,j})) \cdot \delta,$$

where δ is a chosen parameter, h is the height of the hierarchy tree and $l(a_{i,j})$ is the level of the common ancestor $a_{i,j}$. We also have the option of corrupting the similarity matrix by 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 σ. 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

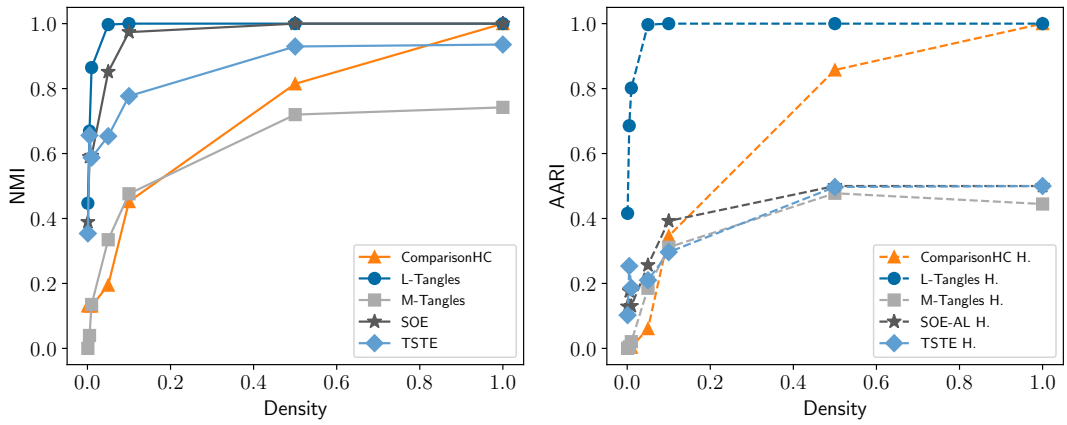
into the hierarchy itself via the noise matrix and the noise that is added to the triplets. The triplet noise has the same form as in Section 4.2. For noise p, every triplet has a chance of p to be flipped. We investigate how our algorithms perform under both noise models, as well as under a lowered density. When evaluating, we compare both the final clustering (the lowest level of the hierarchical block matrix) as well as the obtained hierarchies to each other. To produce a hierarchy with SOE, we apply 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 similarity matrix as well as the triplets randomly.

4.3.2 Lowering density

Setup. As in Subsection 4.2.2, we investigate how each algorithm performs with a lowered density. We draw a differing amount of triplets using the similarities from a hierarchical matrix as described in Subsection 4.3.1, with a density of d indicating that we draw a fraction d of the total amount of triplets. We draw the triplets in a landmark fashion.

Results. The results can be seen in Figure 4.7. We see that L-Tangles performs about on par with SOE in the clustering case (not taking the hierarchy into account), and greatly outperforms all the other algorithms, while M-Tangles and ComparisonHC perform about equally well, with ComparisonHC getting better results in the high triplet case. The performance of t-STE lies in between SOE and ComparisonHC, but we can again see that t-STE does not achieve the correct clustering, even with all triplets drawn. In the hierarchical case, all algorithms perform about as well as they did in the clustering case, with exception of the ordinal embedding algorithms (t-STE and SOE), which perform a lot worse.

Discussion. In the clustering case, we see similar behavior to our gaussian experiments in Subsection 4.2.2: SOE and L-Tangles perform about equally well, with L-Tangles having a slight edge over SOE in the low-density regime. However, for the hierarchical clustering, the ordinal embedding algorithms perform a lot worse. We assume that this has something to do with the inductive bias of the algorithms. Aside from t-STE and SOE, all other algorithms assume in some way that the result is a hierarchy, which could give them a slight edge. It is also curious that ComparisonHC, which was specially designed to cluster hierarchies, only achieves good performance when the density is very high (> 0.5).



(a) Comparing clustering **(b)** Comparing hierarchies

Figure 4.7: L-Tangles achieves top performance in both pure clustering as well as hierarchical clustering. We draw the triplets in a landmark format from a hierarchical noise matrix with 4 clusters and 10 data points each. On the left, we see the NMI of the obtained clusterings versus the ground truth, where we only use the last level of the hierarchies to determine clusters. 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 the ordinal embedding algorithms we have applied agglomerative clustering with average linkage to the obtained embedding instead of k-Means.

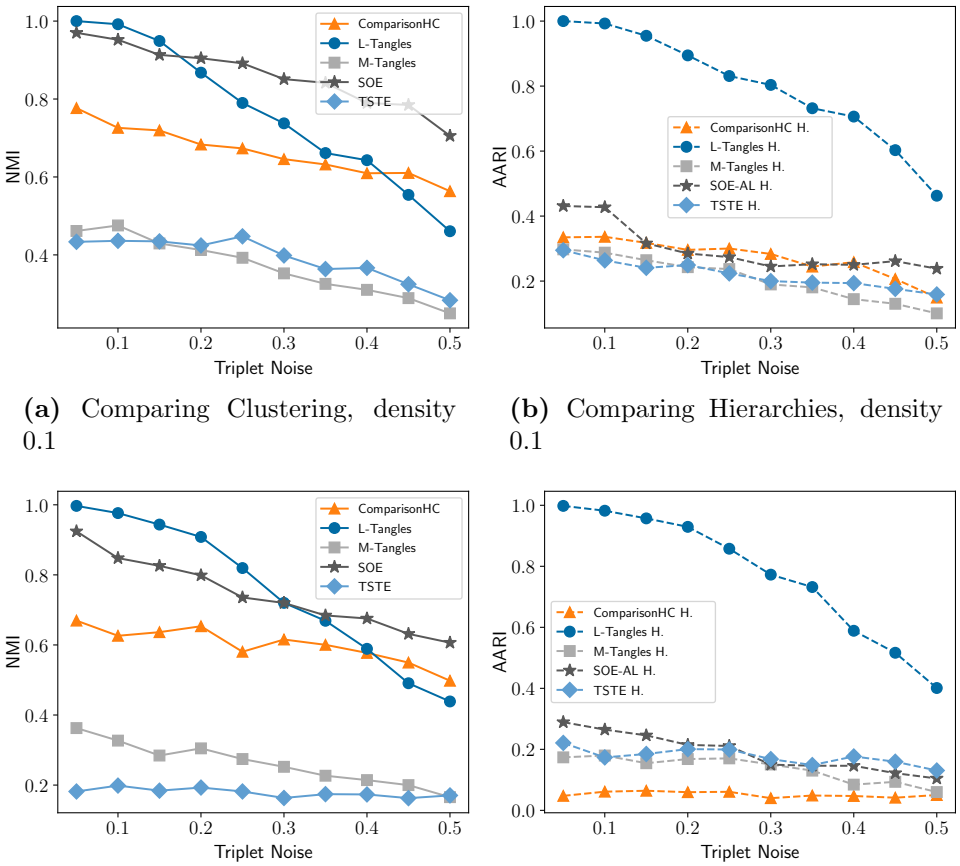
4.3.3 Adding triplet noise

Setup. 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. We repeat the setup under two different densities, with 10% and with 5%, to see if the density has an influence on the noise susceptibility of the algorithms.

Results. The results can be seen in Figure 4.8. First, we look at a pure clustering. In the 10% density case, SOE outperforms all other methods unless the noise is low (< 0.2), where L-Tangles perform 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, t-STE and M-Tangles for both densities and all noise levels. For the hierarchy, L-Tangles greatly outperform all other methods for both densities, with the other methods performing on about 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.

Discussion. We can again see similar behavior to previous experiments. As

in Subsection 4.2.3, L-Tangles fall off more quickly in performance than SOE when adding noise. Additionally, as in Subsection 4.3.2, SOE falls off sharply in performance when clustering hierarchies. In Subsection 4.3.2, we have also noted that the performance of ComparisonHC seems to strongly depend on the number of triplets available. We have observed the same behavior in this experiment, with ComparisonHC performing very poorly for density $d = 0.05$, but on the level of the other algorithms for $d = 0.1$. The other algorithms were not as dependent on the number of triplets available and showed roughly the same performance for both densities.



(a) Comparing Clustering, density 0.1

(b) Comparing Hierarchies, density 0.1

(c) Comparing Clustering, density 0.05

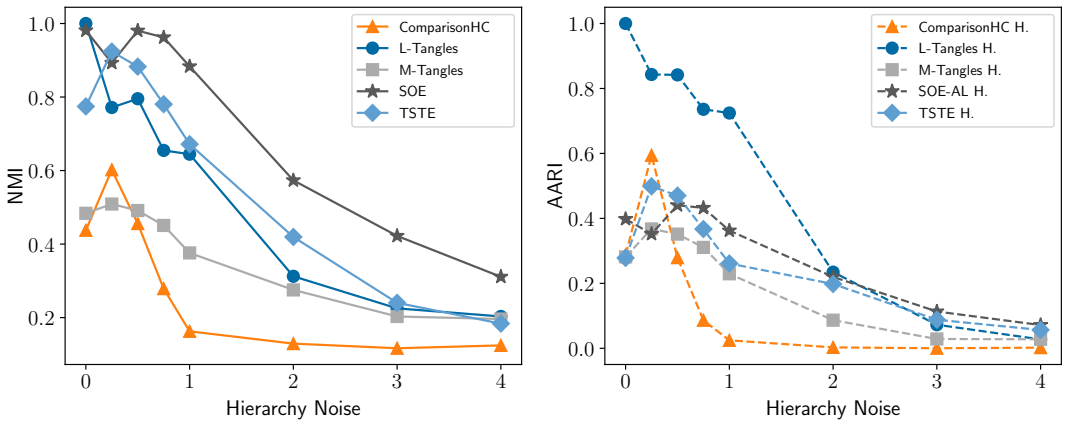
(d) Comparing Hierarchies, density 0.05

Figure 4.8: L-Tangles again performs worse on high-noise data but beats out other algorithms for clustering hierarchies. We plot the performance of different algorithms against the noise introduced on the triplets. If we have a noise of 0.1, we flip a triplet with a probability of 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 draw 10% of them. We draw 20 data points from each cluster in the left column and 200 data points in the right one.

4.3.4 Adding hierarchy noise

Setup. 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 σ^2. We set σ^2 to various values and evaluate the performance of our algorithms.

Results. The results can be seen in Figure 4.9. 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, t-STE 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 also note a curious increase in performance from the lowest noise level to the second lowest noise level in both ComparisonHC as well as t-STE.



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

Figure 4.9: 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 corrupted 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.

Discussion. We see multiple interesting effects here: first, SOE and L-Tangles perform equally well for clustering hierarchies after a certain noise level, when previously L-Tangles always outperformed SOE. 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.

Second, we can see a performance spike both in t-STE as well as ComparisonHC when adding a bit of hierarchy noise. If no noise is present, all points in the same cluster have exactly the same similarities to all other points. When we now for example take three points x_i, x_j, x_k from the same cluster, both (x_i, x_j, x_k) and (x_i, x_k, x_j) would be valid triplets. Our implementation decides which triplet to use based on the index of the data points. We suspect that t-STE and ComparisonHC have a problem with this, and adding a bit of noise can alleviate the issue.

Third, L-Tangles show 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. Even when adding a vanishingly small amount of noise (say 10^{-6}) we see the same steep decline in performance. This illustrates a potential shortcoming of the tangles algorithm, which we will now elaborate on.

Tangles and noise models. First, we want to get some intuition about the hierarchical model. As a visualization aid, we plot a representation of the hierarchical model in Figure 4.10. Care should be taken: this does not accurately visualize 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 with 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 right point in the right set) we receive a *coarse* cut, that divides a higher level of the hierarchy (roughly into left and right). As we see in Figure 4.11a and Figure 4.11b, 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, and 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 the 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, and all cuts from the top-left cluster will be in the right set. Let now c be a point from

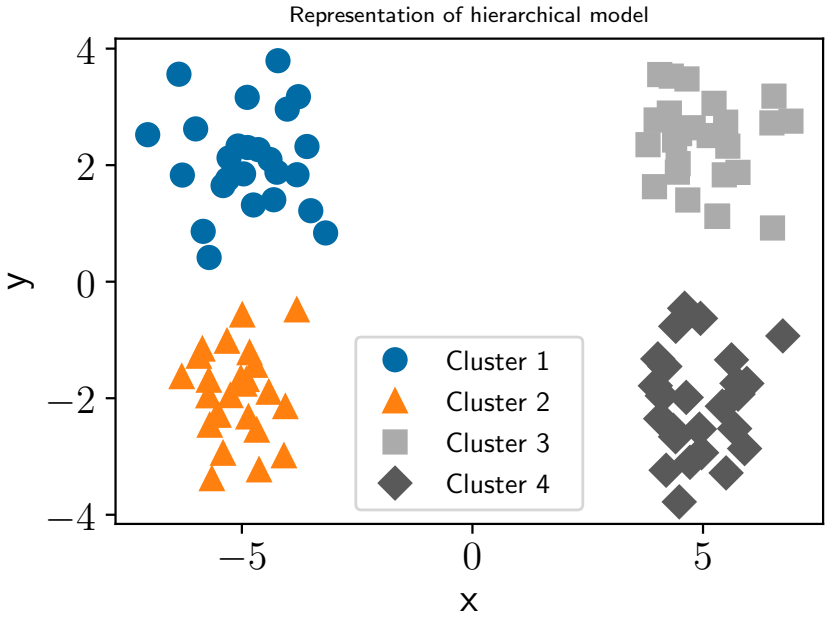


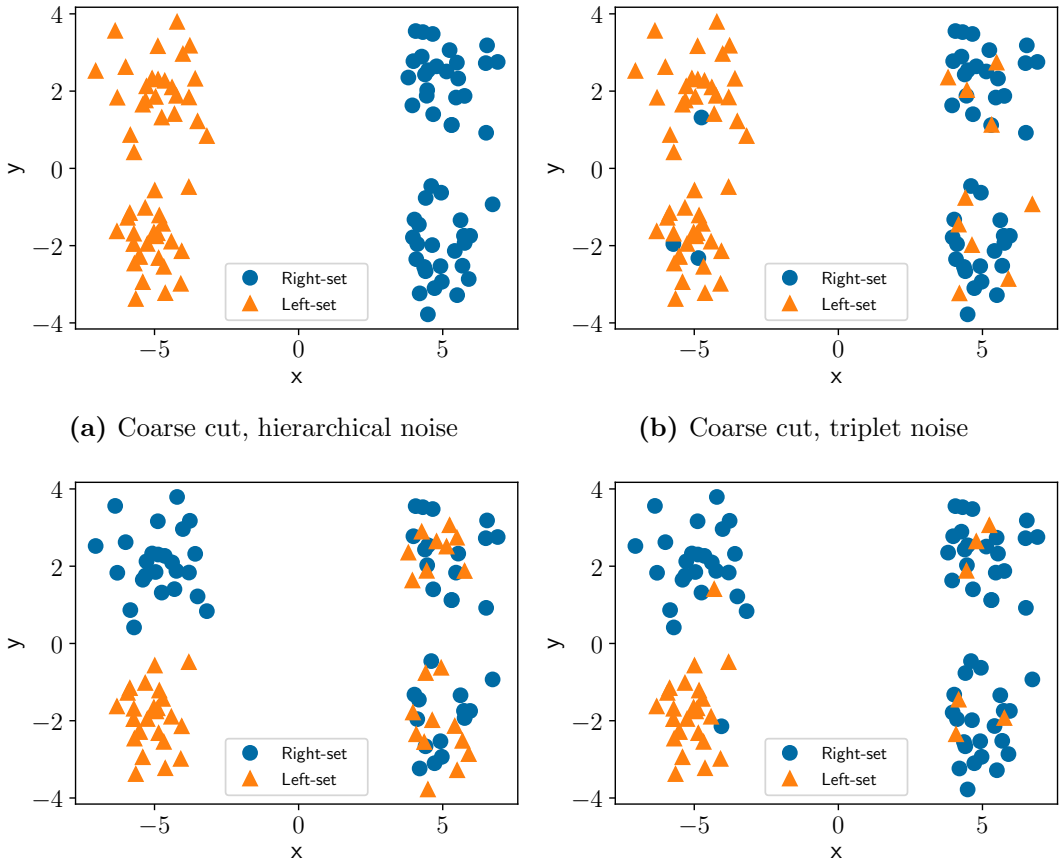
Figure 4.10: A euclidean representation of the hierarchical model with a few more data points drawn. We can sort of see a hierarchy between the clusters, where the left and right sides are two far removed clusters, which can be subdivided into bottom-left, top-left and bottom-right, and top-right. Keep in mind that this representation is only a visualization aid and does not accurately reflect the actual similarity between data points.

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 Land_{ab} only if $d(c, a) < d(c, b)$. Thus, c will not be contained in (a, b). As a result, the set 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 one 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 Land_{ab} (or not). This can also be seen in Figure 4.11c. 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 4.11d.

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 the 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 into 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 oriented in one direction in the left subtree and the other direction in the other subtree. We end up with the correct amount of clusters, and the misclassification will only be a few random points that are assigned to a wrong cluster due to triplet noise.



(a) Coarse cut, hierarchical noise

(b) Coarse cut, triplet noise

(c) Fine cut, hierarchical noise

(d) Fine cut, triplet noise

Figure 4.11: Visualizations of the cuts we receive under both noise models of the hierarchical setting, analog to Figure 4.10. Hierarchical noise means noise that we add directly to the hierarchical similarity matrix, while triplet noise means the percent of triplets answered wrongly. We assume landmark cuts. The coarse cuts are those that separate 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.

Chapter 5

Real World Data from Psychophysics

Psychophysics is a field of study that investigates the influence of physical stimuli on human perception. An example would be the relationship between the wavelength of light and the color sensation that the light produces, see Shepard (1962). To investigate this relationship, researchers in psychophysics often set up experiments to collect triplets from human participants. For the example of color perception, a researcher can place a participant in front of a computer screen, and repeatedly show them three different colors, a, b, c, together with the question *is a more similar to b or c?* The answers of the participants then constitute a triplet data set, which is often analyzed using ordinal embeddings afterward.

In this chapter, we choose a data set consisting of triplets from psychophysics and analyze it using tangles. We compare the results of our analysis with the results that other researchers have achieved using more established methods in psychophysics, such as ordinal embeddings. This chapter thus shows the strengths and weaknesses of tangles on real experimental data.

5.1 Data background

The data we use was collected by Schönmann during her bachelor thesis in Schönmann (2021). 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 questions and image sets. The resulting data sets were then analyzed to show differing similarity perceptions of the participants depending on how the triplet questions were formulated and which image set was used. To collect the data, the participants were presented with three images randomly drawn from the specific set of images, together with one out of four different

questions.

The possible questions are: *Which is the odd one out?*, *Which one is more similar?*, *Which one looks more similar?* and *Which concept is more similar?*, out of which we only study the question *Which is the odd one out?* The images that are shown to the participants come from three different image sets: *action*, *taxonomic* and *thematic*, out of which we only use the *thematic* image set. It consists of 5 classes which can be roughly divided into two themes: barn (straw, hay, pitchforks) and kitchen (forks, dishwashers).

The responses of the participants are converted to triplets as follows: if the participant is presented with images a, b, c and signals that a is the odd one out, we know that b must be more similar to c than to a and c must be more similar to b than to a. Thus, we can gain two triplets from this answer: (b, c, a) and (c, b, a). For each combination of a participant, image set, and question, the data set consists of 462 unique triplets. Participant 2 repeated the experiment over a month later, but we have decided not to include those triplets to stay faithful to the original evaluation.

The data set is particularly suited for our tangles experiment, as the image set consists of images from different classes (for example straw, hay, et cetera). We expect that this makes the resulting triplets particularly suited for clustering. Additionally, the analysis done by Schönmann already identified certain clusters in the data, which we can expect to see in the tangles analysis as well.

5.2 Applying Tangles

5.2.1 Setup

In this section, we show how to apply tangles to the triplet data set of Schönmann. As there is a lot of data present, it is not feasible to repeat our evaluations for all possible data points. We choose the triplets of participant 2 to step through rigorously, and then briefly repeat our evaluations for participant 3. We expect similar results for both observers, as we keep the question and the image set the same.

For the image set, we use the *thematic* one, as Schönmann has reported embeddings that can be cleanly separated into different categories. We also use the *odd-one-out* triplets, as these have been reported in the work by Schönmann as having the highest embedding accuracy. As the triplets are not in a landmark format, but uniformly sampled from the set of all triplets, we use majority cuts.

We process the triplets to cuts using the majority cuts approach with a radius of 1 and apply tangles using an agreement of 3 and with the mean manhattan cost function (see Equation 17 in Section 4.1). As a visualization

aid, we plot the clustering onto a 2d-embedding from SOE. Care must be taken: the embedding from SOE is not a ground-truth embedding and just serves as a visualization aid. If the elements of a cluster are far apart in this visualization, this does not mean that the clustering is wrong – it could just as well be that the distances in the embedding do not correctly represent the ground truth similarity of the data points.

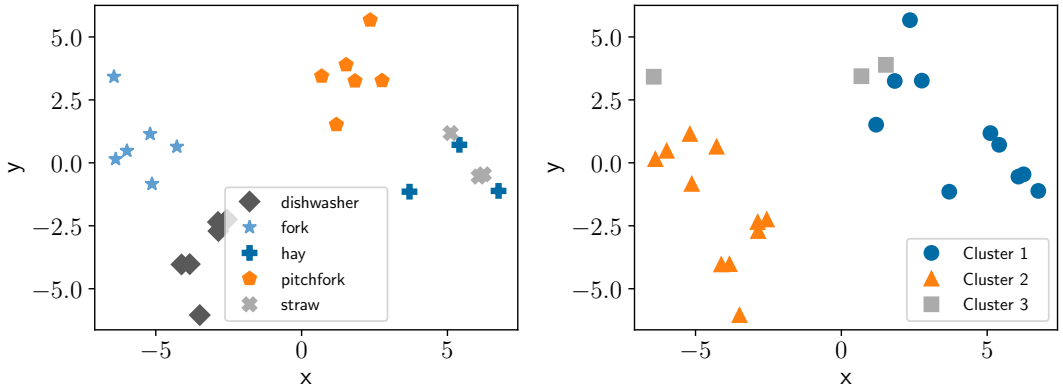
In the analysis, Schönmann qualitatively identified different clusters for participant 2 on the *thematic* image set with the *odd-one-out* triplets. To do this, she identified a separating line in the SOE embedding of the triplets, which divided the objects into the two categories *barn* (hay, pitchfork, straw) and *kitchen* (dishwasher, fork). We expect to see a similar divide in the clusters produced by tangles. As a follow-up, we use the hierarchical clusterings and the explanations produced by tangles to get further insights into the produced clustering.

5.2.2 Embedding and clustering

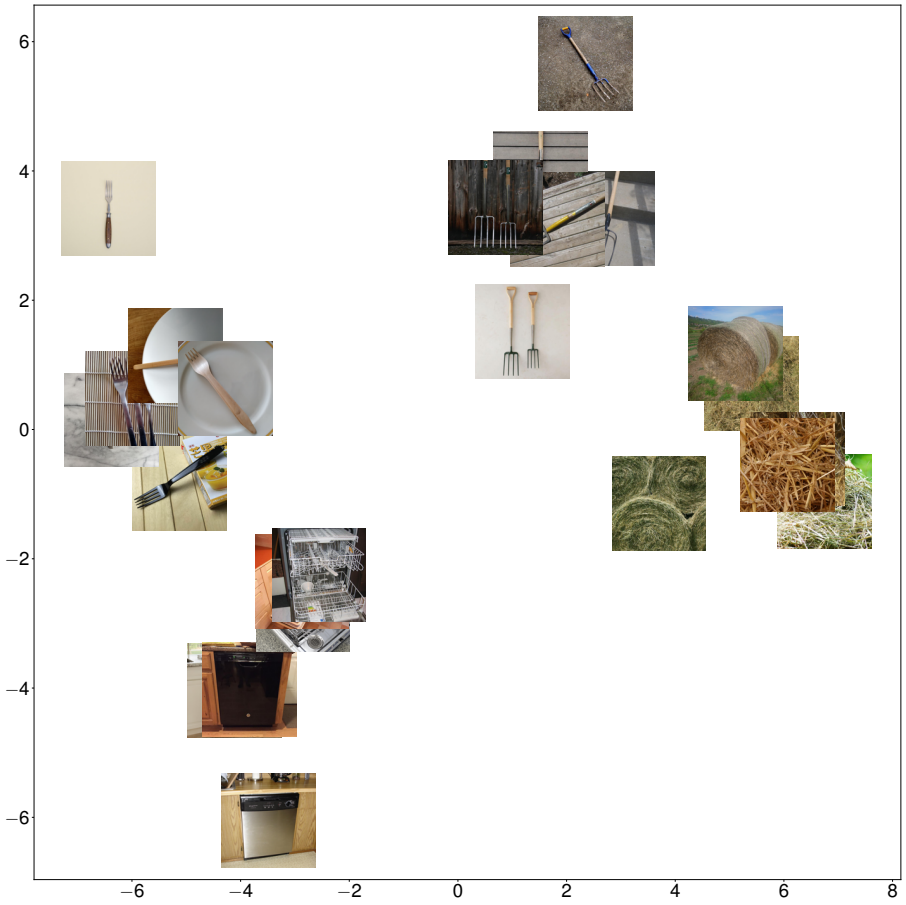
We first reproduce the results by Schönmann by embedding the data with SOE into two dimensions (see Figure 5.1a). In this embedding, one can linearly separate two sets of clusters, which correspond to a divide between kitchen objects (dishwasher, fork) and barn objects (hay, straw, pitchfork), as reported by Schönmann in her thesis. Then we process the triplets to majority cuts, apply tangles and obtain a clustering. These cluster labels are visualised in Figure 5.1b) on the embedding from SOE.

We can see that the clustering from tangles (Figure 5.1b) 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 pitchforks and a kitchen fork. We want to determine whether this is an erroneous clustering, which might arise from too few triplets sampled, or possibly a new insight that we gained.

When we look at items in the grey squares cluster (depicted in Figure 5.2a), we 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 pitchforks, and the two pitchforks look cleaner 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 other than to their counterparts in the kitchen and barn cluster and thus get put into a separate cluster. Interestingly, that is an insight that could not be reached from the SOE embedding alone, as the items are relatively far away from each other in the embedding. Thus, tangles might provide valuable information for a researcher.



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



(c) Embedding with images

Figure 5.1: Embedding of the *odd-one-out* triplets from participant 2 on the thematic image set. In a), we see the original classes, and in b) we see the predictions that we receive by applying majority tangles with an agreement of 3 and a radius of 1. In c), we plotted the original images at the coordinates of their respective SOE embedding.

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

(b) Other kitchen forks

(c) Other pitchforks

Figure 5.2: The images of all forks that are present in the thematic data set. In a), we show the pitchforks 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 (orange triangle cluster, kitchen items) and in c) we depict all other pitchforks (blue circle cluster, barn items).

5.2.3 Hierarchical clustering

Next, we explore what the hierarchy of the clusters looks like. For this, we plot the hierarchy that we receive from the tangles algorithm in Figure 5.3. As expected, we first see a coarse, thematic divide between the kitchen and the barn cluster, with the clean-looking pitchforks being placed together with the kitchen cluster. We then see a finer clustering of the kitchen cluster (node 2), which is split into a set of various kitchen items (node 5), and the cluster of special forks (node 6). The fact that the special forks belong to the coarse kitchen cluster and only get separated in a later step could be interpreted as the participant thinking of the clean-looking pitchforks as belonging more to a kitchen than a barn. This is an insight that we didn't see from the ordinal embedding alone, highlighting a possible strength of tangles.

5.2.4 Explainability

So far we have inspected the clusters that tangles produced. Now, we show how to 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, these are the cuts at the splitting nodes that contribute to a meaningful decision between the left and the right subtree.

We visualize the characterizing cuts in Figure 5.4, together with the images that induce 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 5.4a, the items in the barn cluster are there because they are similar to the two straw/hay images shown in Figure 5.4b.

Our interpretation is that the items that are in the cluster of other kitchen items are there because they are similar to the dishwashers we have plotted in Figure 5.4d. This also makes sense, as we would interpret the dishwashers to very clearly belong in a kitchen environment, while the forks might be more ambiguous.

Overall, there is a 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 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 5.4d is not a strong argument. For example, we would rather know that they are similar to a certain item. To remedy this, we can use more interpretable cuts. If the data would have been suitable for landmark cuts, 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.

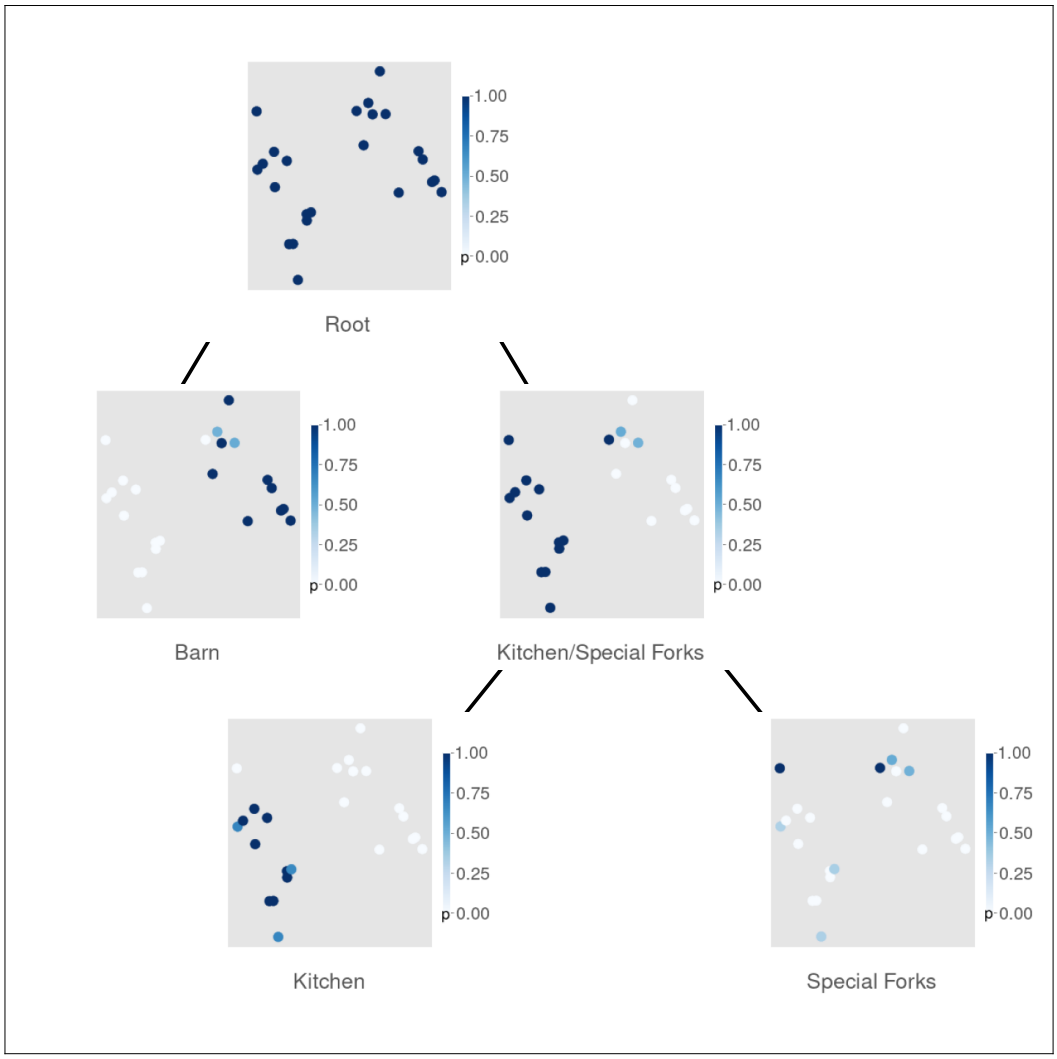
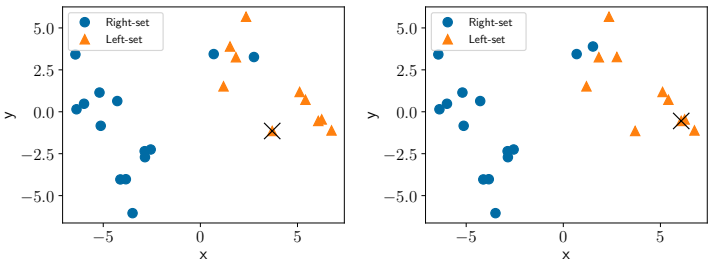
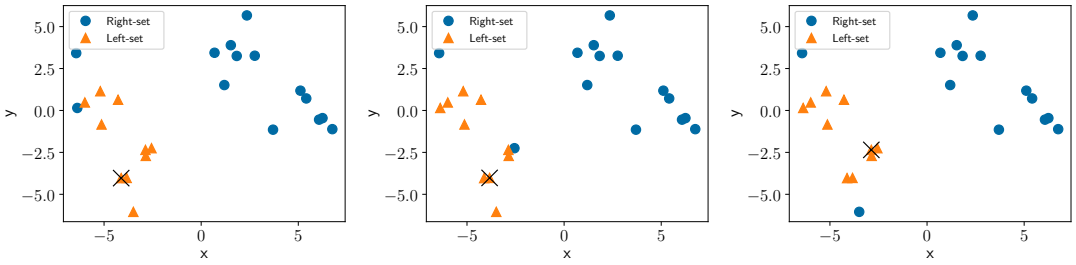


Figure 5.3: The hierarchy produced by the tangles algorithm on the thematic image set for participant 2. Each node represents a (soft) clustering, which we plot in its place. The points are embedded via SOE as a visualization aid. The color of a point corresponds to the probability that the point belongs to the given cluster, the darker, the more probable.



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

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



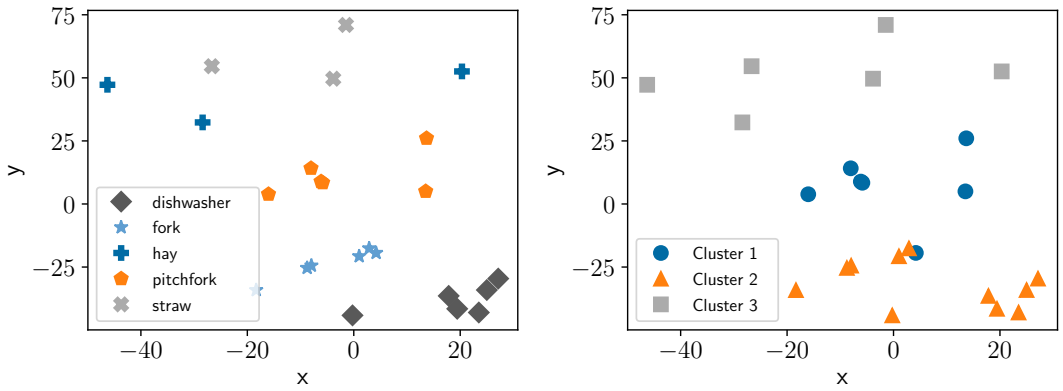
(c) Characterising cuts fine cluster (kitchen/special forks)

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

Figure 5.4: Depiction of the characterizing cuts of all splitting nodes on the thematic image data set. We draw the orientation that corresponds to the left subtree (which is inverted for the right subtree). This means, that 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 characterizing cuts for the root node and in c) have plotted the characterizing cuts for the kitchen/special forks cluster. As these cuts come from our set of majority cuts, we have marked the data point that induced the particular cut with a black X. Below the characterizing cuts in b) and d), we have plotted the images corresponding to the marked points.

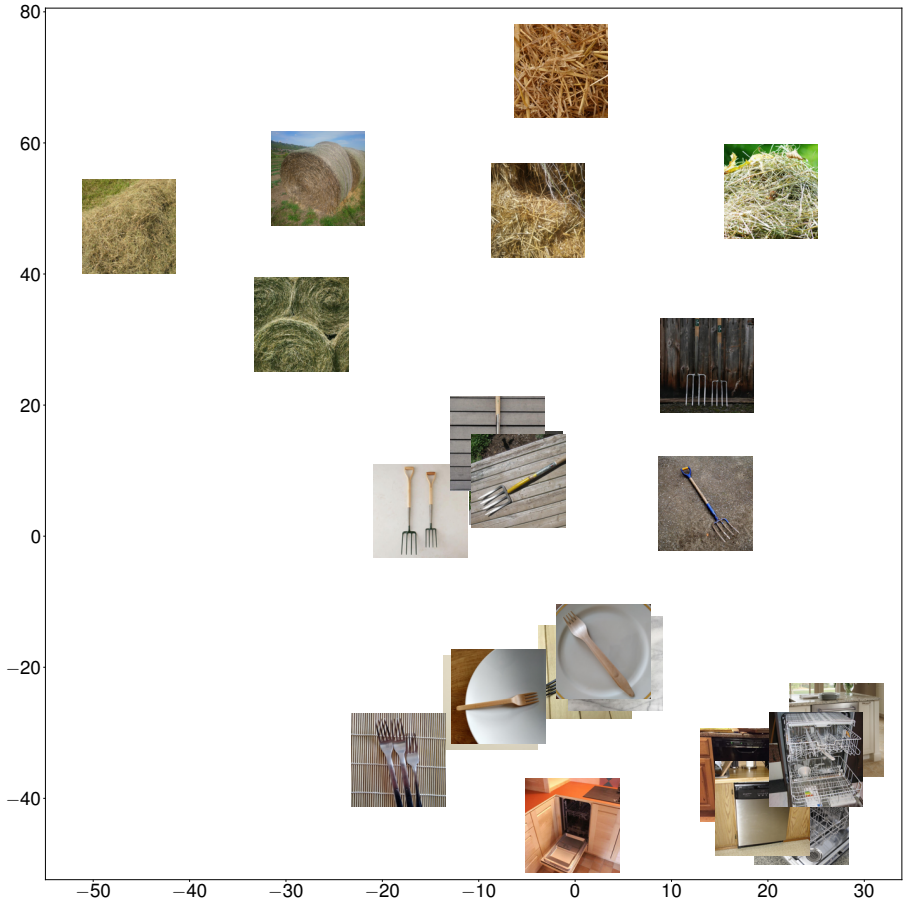
5.2.5 Evaluation of another participant

Next, we check if the results can be repeated for another participant. We select the odd triplets generated from the thematic image set of participant 3 and expect to see similar behavior to our evaluations for participant 2. In Figure 5.5 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 misclassification. If we look at the hierarchy in Figure 5.6, we see that (contrary to participant 2) the pitchforks first get split off, and then the kitchen from the straw-hay cluster. This could indicate that participant 3 deems the straw/hay to be more similar to the kitchen items than to the pitchforks, which could provide valuable insights for further analysis.



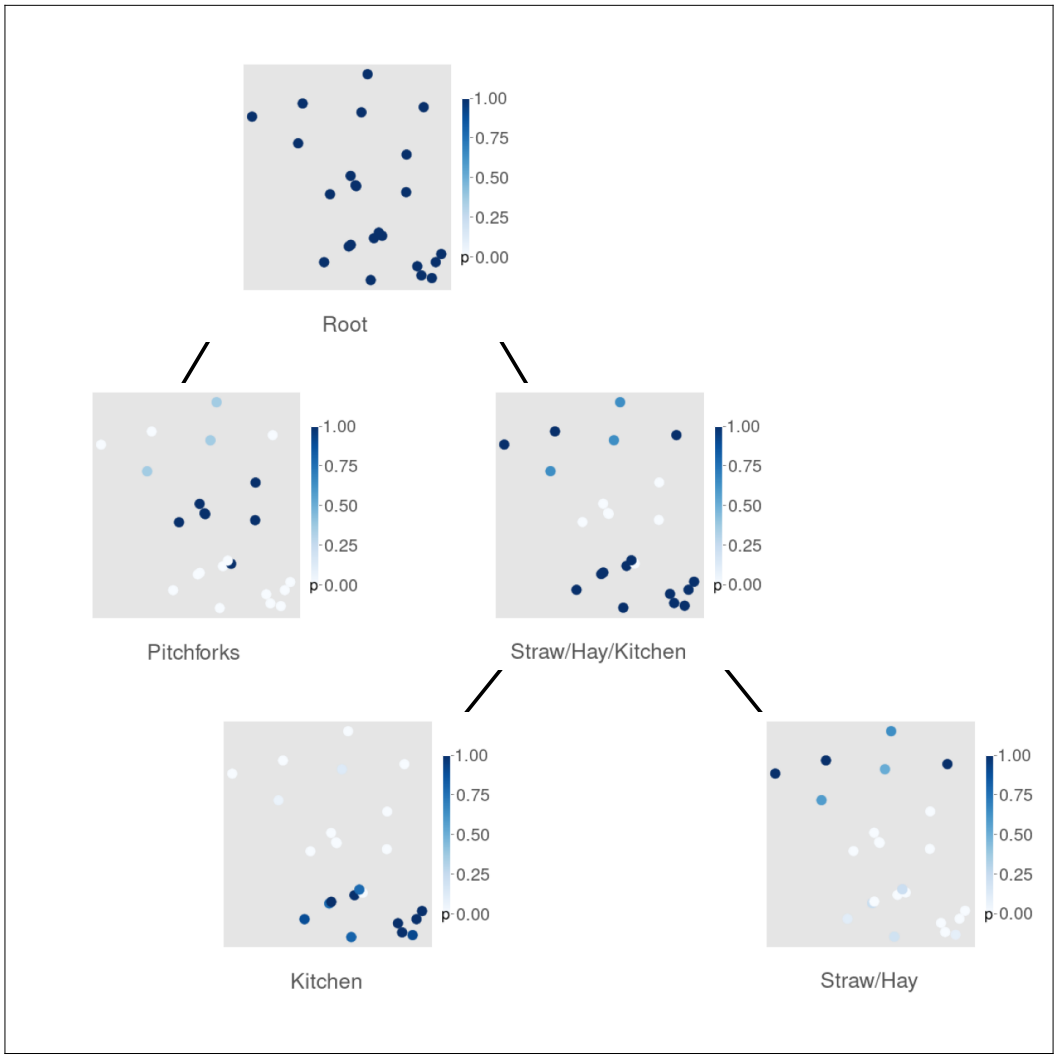
(a) Original classes

(b) Tangles clusters



(c) Embedding with images

Figure 5.5: Analog to Figure 5.1. In a), we plot the original classes over an SOE embedding, in b) we plot the prediction of majority tangles with agreement 3 and radius 1 over a 2-dimensional SOE embedding. In c), we see the original images at the coordinates of their respective SOE embedding.



(a)

Figure 5.6: Tangles produce similar clusters on participant 3 and participant 2. We plot the hierarchy of a tangles clustering on the triplets from participant 3 on the thematic image set. On each node, we plot the soft clustering corresponding to it.

5.3 Discussion

In this chapter, we evaluated tangles on real-world data. We clustered the data with majority tangles, which agreed well with the qualitative analysis that Schönmann has done on the data using more established methods. In addition, the more flexible clustering by tangles can show new dependencies between data points, as we made out a cluster of forks being perceived differently by participant 2.

The hierarchy provided by tangles can help put these new dependencies into a better perspective, and allowed us to gather that the pitchforks in the special fork cluster were perceived to belong more in a kitchen setting than in a barn setting. The explanations by tangles were then used to discover 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 on triplets in psychophysics to this date, allowing tangles to fill a potential niche.

Chapter 6

Conclusion

In this work, we demonstrated a suitable extension for the tangles algorithm to apply it to clustering and hierarchy reconstruction of triplets. This was done via two different methods of generating cuts, *majority cuts* and *landmark cuts*. We validated their properties on simulated and experimental triplet data. As we compared different algorithms in our experiments, we can also give recommendations as to when to use which clustering algorithm and which conditions on the data have to be met for tangles to be effective.

First off, if the goal is explainability, tangles can be a great choice. The importance of explainable algorithms is likely to increase, especially with a *right to explanation* shifting more into the focus of policymakers and legal scholars (Selbst and Powles, 2017), thus tangles could provide a substantial benefit over other algorithms for clustering triplets. To our knowledge, there is no other clustering algorithm that both works with triplet data directly and is explainable. 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.

For standard clustering, we have to differ between the format of the triplets: sampled in a landmark fashion or uniformly at random. For triplets in a landmark fashion, we observed a very good overall performance of landmark tangles. 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. An experimenter that designs a new experiment, 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. For triplets that are sampled uniformly at random, 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 perform best out of our evaluated algorithms. Majority tangles showed the second-best performance, roughly on par with SOE combined with average linkage. It can be problematic that tangles cannot construct a true dendrogram, which is what the hierarchical clustering literature focuses on (such as in Ghoshdastidar et al. (2019)). For practical applications, however, the output from tangles can be good enough.

In the setting of psychophysics, we have received good practical results with tangles on majority cuts, confirming and expanding on the insights that the original author had gained through an ordinal embedding. Although, we think that tangles could perform better under different circumstances. As the analyzed data set consisted of uniformly sampled triplets, we had to use majority tangles in our analysis. However, in our simulations, landmark tangles performed much better than majority tangles on all data sets. Also, as discussed in Subsection 5.2.4, landmark tangles can provide explanations that are more intuitive than majority tangles, as landmark cuts are inherently more explainable.

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 on or modified. As we demonstrated, changing how to preprocess triplets to cuts is an effective method of applying tangles to triplets. 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 on much is the cost function. One could imagine different cost functions than the mean manhattan cost function that we used. An idea could be to estimate a similarity on the data points using the triplets, for example, done in Kleindessner and von Luxburg (2017), or use the average-linkage cluster similarity in Ghoshdastidar et al. (2019) and use this to calculate the cost of a cut. Additionally, one could think about modifying parts of the tangles framework themselves, which could help with the problem of hierarchy noise we encountered in Subsection 4.3.4.

An addition to our research would be applying tangles to a more favorable real-world data set. This means that the data set contains both:

- objects that exhibit a cluster structure, for example, distinct classes, such as the analyzed data set from Schönmann
- triplets sampled in a landmark format

To our knowledge, such a data set did not exist when we conducted our evalu-

ations. However, this data could be readily generated in a controlled environment, for example, using the approach from Schönmann (2021).

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. S Bernstein, and J Heer. Learning Perceptual Kernels for Visualization Design. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1933–1942, 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. *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 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.

Kernighan, Brian Wilson and Lin, Shen. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

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. *2012 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. *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