1 Introduction

This project investigates the methods for clustering of tree data structures. Clustering is an important method of unsupervised learning that allows to discover new properties and patterns within data.

Tree clustering plays an important role for the scientific areas which use tree structures to describe observations, e.g. computational biology, structured text databases, natural language processing, web mining, image analysis and computer vision, pattern recognition as well as compiler optimisation [3, 9, 14, 24]. This work illustrates the application of tree clustering on set of experimental observations of stem cell differentiation.

The main contribution of this work is the development of the methods for finding mean trees in a set of unordered labelled trees. The traditional methods, i.e. calculation of median graph [11], are not as flexible in the terms of using different similarity metrics, as well as would lead to infeasible centroid trees. To the best of our knowledge this project presents the first study of this approach. The related works for different algorithms used in the project are reviewed in the corresponding sections.

The remain of this script starts with the description of different methods to establish a distance between two tree structures in Section 2. This allows to define a metric on a tree space, which is necessary for the correctness of centroid search and clustering algorithms discussed in Sections 3 and 4.

The experimental part of this script in Section 7 begins with evaluation of different metrics and clustering algorithms using artificial data, following by application on the real-world data of biological experiments.

The running time evaluation of algorithms used in this project as well as some methods for optimisation and parallelisation are discussed in Section 8.

The last section of this script summarises the results and gives an overview of possible topics for further research.
2 Labelled Tree Distances

A tree is a tuple $T = (V, E)$, where $V$ is a set of nodes and $E$ is a set of directed edges between the nodes. A node $v$ has a child $w$ if there is an edge $(v, w) \in E$.

The object of interest for this work are unordered labelled trees with root. $T$ is called labelled if a function is defined that maps every node $v$ of the tree to the elements of alphabet an $\Sigma$. The tree is called unordered if the order between two arbitrary sibling nodes is not fixed.

A metric or distance on a set $A$ is a function $d : A \times A \rightarrow \mathbb{R}$. Usually metric describes dissimilarity between two elements of a set. In order to be a proper metric, $d$ has to fulfil following properties $\forall x, y, z \in A$:

**non-negativity:** $d(x, y) \geq 0$

**identity of indiscernibles:** $d(x, y) = 0 \iff x = y$

**symmetry:** $d(x, y) = d(y, x)$

**triangle inequality** $d(x, z) \leq d(x, y) + d(y, z)$

2.1 Constrained Tree Edit Distance

Unfortunately, the scientific community does not agree on one established method to find a metric between trees. One commonly used method is Tree Edit Distance (TED) [25]. Similar to Levenstein edit distance, tree edit distance is defined as the minimal number of operations needed to transform one tree into another. The allowed operations are creation, removing and renaming of a node.

Arora et al. showed that for unordered Labelled trees the calculation of TED is NP-hard and even MAX SNP-hard [2]. And while for ordered Labelled trees there are different methods to apply dynamic programming, the application of these methods to unordered trees would lead to a considerable loss of efficiency.

Zhang suggested to use constrained TED ($cTED$) to calculate metrics for unordered trees. This solves the problem in $O(|T_1| \times |T_2| \times (\deg(T_1) + \deg(T_2)) \times \log_2(\deg(T_1) + \deg(T_2)))$ [23].

Constrained edit distance mapping is defined by a triple $(M, T_1, T_2)$, where $T_1$ and $T_2$ are two trees and $M$ is a set of ordered tuples $(v, w) \in V_1 \times V_2$ which satisfies the following conditions:

1. $M$ is an edit distance mapping

2. $\forall (v_1, w_1), (v_2, w_2), (v_3, w_3) \in M$ let $T_1[v] := \text{lca}(T_1[v_1], T_1[v_2])$ and $T_2[w] := \text{lca}(T_2[w_1], T_2[w_2])$, where lca represents the least common ancestor and $T[v]$ represents a subtree of $T$ induced by a node $v$. $T_1[v]$ is a proper ancestor of $T_1[v_3]$ iff $T_2[w]$ is a proper ancestor of $T_2[w_3]$.
The first condition means that $M$ should injectively map nodes of $T_1$ to nodes of $T_2$ maintaining an ancestor-descendant relationship between the mapped nodes.

The second condition ensures that two different subtrees of $T_1$ have to be mapped on two different subtrees of $T_2$. This condition is sufficient and even desirable for many different problems, in particular for the problem discussed later in this script, where nodes represent the phases of cell separation. Clustering would have to discover similar separation patterns and transition of cells through similar phases. Here we want to avoid the cases where two different alternatives of behaviour in one experiment are mapped to an ancestor and a descendant in another.

**Algorithm**

Algorithms to calculate cTED adapted from the paper by Zhang [23] are shown in Algorithms 1 and 2. Constrained TED is a dynamic programming [16] method that solves a large optimisation problem by breaking it down into smaller sub-problems.

The solutions of the sub-problems are stored in the matrix $D_{TT}[v, w]$ which represents the constrained tree edit distances between (sub)trees with root nodes $v$ and $w$. This distances, once computed, can be reused in different steps of the algorithm to avoid redundant computations. If $D_{TT}[v, w]$ has a non-negative entry, cTED of those two nodes was already computed (line 3). In the description of the algorithm we use the notation $D_{TT}(v, w)$ to denote that the distance is required in the current step and, if it is not yet stored in the matrix, the recursive call of the cTED method will be made.

Besides the distance between two trees the algorithm also stores the edit distance $D_{FF}[v, w]$ between forests $F_1[v]$ and $F_2[w]$. The forest $F_i[j]$ is obtained from $T_i[j]$ by removing the root node $j$ together with all edges to its children.

The algorithm starts with an initialisation phase. Since no operations have to be performed, the costs to transform an empty tree to an empty tree or an empty forest to an empty forest are equal to 0 (line 2). Lines 5 to 13 compute the costs to remove the subtrees and forests rooted on every node of the tree $T_1$. Lines 14 to 22 compute the costs of creating every subtree and forest rooted on every node of the tree $T_2$. These values are then used in the second part shown in Algorithm 2.

The fundamental idea of the algorithm is that—as was shown by Zhang—the distance between to subtrees $T_1[v]$ and $T_2[w]$ with root nodes $v$ and $w$ correspondingly is equal to the minimum (line 48) between

1. creating $T_2[w]$ from an empty tree $\emptyset$ whereat $T_1[v]$ may be transformed to a subtree $T_2[w_t]$ ($d_4$, lines 37–41): 

$$D_{TT}(\emptyset, T_2[w]) + \min_{w_t \in T_2[w]} \{D_{TT}(T_1[v], T_2[w_t]) - D_{TT}(\emptyset, T_2[w_t])\}, \quad (1)$$
Algorithm 1 Constrained TED Algorithm by Zhang [23], Part 1

1: function cTED($T_1$, $T_2$, $D_{TT}$, $D_{FF}$)  
2: \[ D_{TT}[\emptyset, \emptyset] = D_{FF}[\emptyset, \emptyset] = 0 \]
3: if $D_{TT}[T_1, T_2] \neq -1$ then return $D_{TT}[T_1, T_2]$  
4: end if
5: for all node $\in T_1$ do  
6: \[ d = 0 \]
7: for all c a child of node do  
8: \[ D_{TT}[c, \emptyset] = cTED(c, \emptyset, D_{TT}, D_{FF}) \]
9: \[ d+ = D_{TT}[c, \emptyset] \]
10: end for
11: \[ D_{FF}[\text{node}, \emptyset] = d \]
12: \[ D_{TT}[\text{node}, \emptyset] = D_{FF}[\text{node}, \emptyset] + \text{cost}_{\text{remove}} \]
13: end for
14: for all node $\in T_2$ do  
15: \[ d = 0 \]
16: for all c a child of node do  
17: \[ D_{TT}[\emptyset, c] = cTED(\emptyset, c, D_{TT}, D_{FF}) \]
18: \[ d+ = D_{TT}[\emptyset, c] \]
19: end for
20: \[ D_{FF}[\emptyset, \text{node}] = d \]
21: \[ D_{TT}[\emptyset, \text{node}] = D_{FF}[\emptyset, \text{node}] + \text{cost}_{\text{add}} \]
22: end for
2. removing $T_1[v]$ with exception of a subtree $T_1[v_s]$ which will be transformed to $T_2[w]$ ($d_5$, lines 42–46):
\[
D_{TT}(T_1[v], \emptyset) + \min_{v_s \in F_1[v]} \{D_{TT}(T_1[v_s], T_2[w]) - D_{TT}(T_1[v_s], \emptyset)\}, \tag{2}
\]

3. transforming the forest of subtrees $F_1[v]$ induced by children of $v$ to the forest of subtrees $F_2[w]$ induced by children of $w$ and relabelling $v$ into $w$ ($d_6$, line 47):
\[
D_{FF}(F_1[v], F_2[w]) + \text{cost}_{\text{relabel}}. \tag{3}
\]

Furthermore, the distance between two induced forests $D_{FF}(F_1[v], F_2[w])$ is equal to the minimum (line 35) of the costs to

1. create the forest $F_2[w]$ from an empty set and using $F_1[v]$ as a part forest ($d_1$, line 32):
\[
D_{FF}(\emptyset, F_2[w]) + \min_{w_t \in F_2[w]} D_{FF}(F_1[v], F_2[w]) - D_{FF}(\emptyset, F_2[w_t]), \tag{4}
\]

2. remove the forest $F_1[v]$ with exception of subforest $F_1[v_s]$ which will be transformed to $F_2[w]$ ($d_2$, line 33)
\[
D_{FF}(F_1[v], \emptyset) + \min_{v_s \in F_1[v]} \{D_{FF}(F_1[v_s], F_2[w]) - D_{FF}(F_1[v_s], \emptyset)\}, \tag{5}
\]

3. and the solution of minimum cost maximum flow problem on the graph $G(v, w)$ created from $F_1[v]$ and $F_2[w]$ (see [23] for details) ($d_3$ line 34).

The solution of the minimum cost maximum flow problem is very well studied and described in the literature (see, for example, [11]) and therefore will not be described in this script in detail.

2.2 Maximal Similarity Common Subtree

Another suitable method to establish a metric for the space of unordered labelled trees was suggested by Torsello et al. [21]. This method is based on the computation of maximal similarity common (MaxSimilarity) subtree between two trees.

Two trees $T_1$ and $T_2$ are called isomorphic if there is an isomorphism $\phi$ that maps each node of the tree $T_1$ to each node of the tree $T_2$.

For two subtrees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ the bijection $\phi : H_1 \rightarrow H_2$, with $H_1 \subseteq V_1$, $H_2 \subseteq V_2$ is called subtree isomorphism if:

1. $\forall u, v \in H_1 : u$ adjacent with $v \iff \phi(u)$ adjacent with $\phi(v)$ and
2. both induced subtrees $T_1[H_1]$ and $T_2[H_2]$ are connected
Algorithm 2 Constrained TED Algorithm by Zhang [23], Part 2

23: for all \( v \in T_1 \) do
24:    for all \( w \in T_2 \) do
25:       if \( v \) is a leaf and \( w \) is a leaf then
26:          \( D_{FF}[v, w] = D_{FF}[^{\emptyset}, ^{\emptyset}] \)
27:       else if \( v \) is a leaf and \( w \) is not a leaf then
28:          \( D_{FF}[v, w] = D_{FF}[^{\emptyset}, w] \)
29:       else if \( v \) is not a leaf and \( w \) is a leaf then
30:          \( D_{FF}[v, w] = D_{FF}[v, ^{\emptyset}] \)
31:       else
32:          \( d_1 = D_{FF}[^{\emptyset}, w] + \min\{D_{FF}[v, k] - D_{FF}[^{\emptyset}, k] \mid \forall k \text{ child of } w\} \)
33:          \( d_2 = D_{FF}[v, ^{\emptyset}] + \min\{D_{FF}[k, w] - D_{FF}[k, ^{\emptyset}] \mid \forall k \text{ child of } v\} \)
34:          \( d_3 = \text{MINMM}(v, w, D_{TT}, D_{FF}) \)
35:          \( D_{FF}[v, w] = \min\{d_1, d_2, d_3\} \)
36:       end if
37:       if \( w \) not a leaf then
38:          \( d_4 = \text{cTED}[^{\emptyset}, w, D_{TT}, D_{FF}] + \\min\{\text{cTED}(v, k, D_{TT}, D_{FF}) - \text{cTED}(^{\emptyset}, k, D_{TT}, D_{FF}) \mid \forall k \text{ child of } w\} \)
39:       else
40:          \( d_4 = \text{cTED}[^{\emptyset}, w, D_{TT}, D_{FF}] + \text{cTED}(v, ^{\emptyset}, D_{TT}, D_{FF}) \)
41:       end if
42:       if \( v \) not a leaf then
43:          \( d_5 = \text{cTED}(v, ^{\emptyset}, D_{TT}, D_{FF}) + \\min\{\text{cTED}(k, w, D_{TT}, D_{FF}) - \text{cTED}(k, ^{\emptyset}, D_{TT}, D_{FF}) \mid \forall k \text{ child of } v\} \)
44:       else
45:          \( d_5 = \text{cTED}(v, ^{\emptyset}, D_{TT}, D_{FF}) + \text{cTED}(^{\emptyset}, w, D_{TT}, D_{FF}) \)
46:       end if
47:       \( d_6 = D_{FF}[v, w] + \text{cost}_{\text{relabel}} \)
48:       \( D_{TT}[v, w] = \min\{d_4, d_5, d_6\} \)
49:    end for
50: end for
51: end function
Problem: Find maximum similarity subtree isomorphism $\phi$, so that $W_\sigma(\phi) = \sum_{u \in H_1} \sigma(u, \phi(u))$ is the largest among all subtree isomorphisms between $T_1$ and $T_2$.

Let $\sigma(u, w)$ be the similarity function. Then the common similarity between subtrees $T_1[H_1]$ and $T_2[H_2]$ is defined as

$$W_\sigma(\phi) = \sum_{u \in H_1} \sigma(u, \phi(u)) \quad (6)$$

and maximal similarity isomorphism $\phi^* = \arg\max_\phi W_\sigma(\phi)$.

In their paper Torsello et al. show that the following functions are proper metrics:

$$d_1(T_1, T_2) = \max(|T_1|, |T_2|) - W_\sigma(\phi^*) \quad (7)$$

$$d_2(T_1, T_2) = |T_1| + |T_2| - 2W_\sigma(\phi^*) \quad (8)$$

$$d_3(T_1, T_2) = 1 - \frac{W_\sigma(\phi^*)}{\max(|T_1|, |T_2|)} \quad (9)$$

$$d_4(T_1, T_2) = 1 - \frac{W_\sigma(\phi^*)}{|T_1| + |T_2| - W_\sigma(\phi^*)} \quad (10)$$

The metrics (9) and (10) are normalised on the interval $[0, 1]$, and can be advantageous in some cases [21].

The subtree isomorphism between trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ is called anchored at nodes $v \in V_1$ and $w \in V_2$ if the subtrees of $T_1$ and $T_2$ induced by the isomorphism are rooted at $v$ and $w$. In this case the maximum similarity isomorphism anchored at $v$ and $w$ is equal to similarity between $v$ and $w$ and maximal similarity between the subtrees induced by the children of nodes $v$ and $w$ (this subtrees are then assigned to be isomorphic):

$$W(\phi(v, w)) = \sigma(v, w) + \max_{\pi \in \Sigma_n^m} \sum_{v_s \text{ child of } v} W(\phi(v_s, \pi(v_s))), \quad (11)$$

where $\Sigma_n^m$ is the space of all possible assignments between a set of cardinality $n$ (the number of children nodes of $v$) and a set of cardinality $m$ (the number of children of $w$).

Algorithm

Algorithm 3 illustrates the routines for calculation of the maximum similarity subtree isomorphism. The function MAXSIMILARITY iteratively computes the subtree isomorphisms anchored at every node of $T_1$ with $T_2$ and vice verse using the function ANCHOREDSIMILARITY. The function ANCHOREDSIMILARITY searches for anchored maximum similarity isomorphism using Formula (11), where the values of maximum similarity isomorphism of subtrees anchored at the children nodes are computed recursively.
The assignment problem is solved in the function Assign using Kuhn-Munkres algorithm [5][18], which is a standard method for finding the maximal assignment and is described in detail in [5].

In our program the similarity function $\sigma(u,w)$ returns 1 if two nodes have the same label and 0 otherwise.

**Algorithm 3** Maximum Similarity Common Subtree Algorithm by Torsello et al. [21]

\begin{verbatim}
function MaxSimilarity(T1, T2)
    maxsim = 0
    for all u ∈ T1 do
        sim = AnchoredSimilarity(u, T2[root])
        if sim > maxsim then
            maxsim = sim
        end if
    end for
    for all w ∈ T2 do
        sim = AnchoredSimilarity(T1[root], w)
        if sim > maxsim then
            maxsim = sim
        end if
    end for
    return maxsim
end function

function AnchoredSimilarity(u, w)
    for all u_i child of u do
        for all w_j child of w do
            w_ij = AnchoredSimilarity(u_i, w_j)
        end for
    end for
    return $\sigma(u, w) + \text{Assign}\{w_{ij}\}$
end function
\end{verbatim}

The difference between the values $\gamma$ for cTED and $\sigma$ for MaxSimilarity is that the larger values of $\gamma$ mean larger differences (distances, dissimilarities) between nodes (usually 1 if nodes have different labels and 0 if same), while $\sigma$ shows similarity (1 if labels are same and 0 if they are different).

Both methods to calculate the distance are compared in Section 7.1.

3 Mean, Median and Medoid Trees

Centroids play a key role in partitional clustering methods discussed in this script. A centroid is an element that minimises the sum of distances between
itself and all elements in the set (also called *sum of stars*) under certain constraints. Dependent on the minimisation problem and constraints we differentiate between three types of centroids: mean, median and medoid.

### 3.1 Median and Mean Trees

We define *mean tree* as the minimiser $\bar{T}$ of the optimisation problem

$$\min_{\bar{T} \in \mathcal{T}} \sum_{i=1}^{n} d(\bar{T}, T_i)^2$$  \hspace{1cm} (12)

over a set of trees $T_1, T_2, \ldots, T_n \in \mathcal{T}$ in the space $\mathcal{T}$, with $d$ the metric function defined as described in the previous section.

Without squaring the distances we obtain a multi-dimensional generalisation of the median of a set of points from $\mathbb{R}$. Therefore, we define *median tree* as the minimiser of the optimisation problem

$$\min_{\bar{T} \in \mathcal{T}} \sum_{i=1}^{n} d(\bar{T}, T_i).$$  \hspace{1cm} (13)

Since we could not see a way to determine a sub-differential on $\mathcal{T}$, we used the random search method [19, 20] to find the minimiser. Algorithm 4 illustrates the application of random search for finding a centroid tree. The function $\text{FindCentroid}$ expects the set of trees $\mathcal{T}$ as input parameter. The algorithm initialises a centroid with some random feasible tree and searches for possible improvements that minimise the sum of distances. In order to succeed the algorithm samples transformations from the space of possible transformations\footnote{Those transformations are relabelling of a node, removing a leaf or creation of new children for a leaf.} of the centroid (method $\text{SampleRandomTransformations}$ in line 4) and verifies if the modified centroid $\text{new}$ has a smaller sum of distances than the incumbent solution. If so, the modified centroid becomes the new incumbent solution (line 7).

If nothing else is mentioned in the description of the experimental setup, the algorithm terminated if 100 tested transformations in a row could not offer any improvement compared to the current state. In order to avoid local minima we repeated the procedure 20 times with different initial trees and took the best solution as the final minimiser.

### 3.2 Medoid Tree

Similarly to the median tree one can define a *medoid tree* by applying the constraint $\bar{T} \in \mathcal{T}$ to (12) or (13). In this project we decided to use medoid trees that minimise (13).
Algorithm 4 FindCentroid(T)

1: centroid = RANDOMTREE( )
2: repeat
3:  testedTransformations = 0
4:  transformation = SAMPLERANDOMTRANSFORMATION(centroid)
5:  centroid<sub>new</sub> = APPLYTRANSFORMATION(centroid, Transformation)
6:  if $\sum_{T \in T} d(T, \text{centroid}<sub>new</sub>) < \sum_{T \in T} d(T, \text{centroid})$ then
7:     centroid = centroid<sub>new</sub>
8:  else
9:     testedTransformations = testedTransformations + 1
10: end if
11: until testedTransformations < 100
12: return centroid

For this class of problems no stochastic optimisation is required. It is enough to compute the distance matrix (dissimilarity matrix) between all points and select the tree with the minimal row sum. This can be implemented very efficiently using the numerical libraries, i.e., NumPy\(^2\) for Python. As it will be shown in Section 4.2, k-medoid clustering can also be implemented in a much more efficient way than the general algorithm.

4 Clustering

There are a number of different methods to partition a set of elements in a space with defined similarity/dissimilarity metric [12, 22]. In this work we focus on k-means and k-medoids clustering algorithms as those are very well known and commonly used methods.

In this work we refer to the k-medians algorithm as a variation of k-means, which uses median trees defined in Section 3.1 for centroids. Some authors call the k-medians algorithm the partitioning, where centroids are elements of the dataset [17]. In the notation adopted in this script we call such clustering method k-medoids in order to avoid misunderstanding.

4.1 Clustering with k-Means and k-Medoids Algorithms

Clustering within the framework of k-means finds a partition of the dataset $A = \{a_1, \ldots, a_N\}$ into disjoint non-empty subsets $B_i, \bigcup_i B_i = A$, together with a set of centroids $c_i$, with $i = 1, \ldots, k$. This partitioning minimises the sum of squared distances between each point $a_j$ and the centroid $c_i$ of the cluster $B_i$ containing it. This can be written as constrained non-linear

\(^2\)http://numpy.scipy.org
optimisation problem:

$$\min E(W, C) := \sum_{i=1}^{k} \sum_{j=1}^{N} w_{ij} d(a_j, c_i)^p$$ (14)

subject to

$$w_{ij} \in \{0, 1\}, \quad \text{for } 1 \leq i \leq k, 1 \leq j \leq N,$$ (15)

$$\sum_{i=1}^{k} w_{ij} = 1 \quad \text{for } 1 \leq i \leq k, 1 \leq j \leq N.$$ (16)

Here $C := \{c_1, \ldots, c_k\}$ are the centroid locations, and $W := (w_{ij})$ is the partition matrix corresponding to the partition $B_i$ of $A$. In the matrix $W$, $w_{ij}$ assumes the value 1 if element $a_j$ is assigned to the cluster $B_i$ and 0 otherwise. Whether k-means or k-medians clustering is performed depends on the value of the power quotient $p$. For $p = 1$ the k-medians and for $p = 2$ the k-means clustering is carried out.

A common approach to minimise (14) subject to (15) and (16) is partial optimisation for $W$ and $C$, i.e. alternating minimisation with respect to either $W$ or $C$ while keeping the other one fixed [4].

The batch $k$-means algorithm employs precisely this strategy: after an initial random choice of centroids $c_1, \ldots, c_k$ it iterates between the following two steps until convergence where neither partitioning nor centroids are changing anymore:

**Cluster assignment:** $\forall a_t \in A$ determine an index $i(t)$ such that

$$i(t) = \arg \min_i d(a_t, c_i)$$ (17)

**Cluster update:** within each cluster $B_i := \{a_t | i(t) = i\}$ determine the centroid $c_i$ by minimising

$$c_i := \arg \min_c \sum_{a \in B_i} d(a, c)^p$$ (18)

Step (17) does not depend on the particular data structure once the dissimilarity metric is defined. Step (18) is performed using combinatorial optimisation methods described in Section 3.1. The algorithm is illustrated on Figure 1.

### 4.2 Clustering with the k-Medoids Algorithm

Similar to (14), (15), and (16) one can formalise the k-medoids problem by applying the additional constraint that centroids should be elements of the dataset:
Figure 1: Flow chart of the k-means algorithm for tree clustering.

\[ \min E(W, C) := \sum_{i=1}^{k} \sum_{j=1}^{N} w_{ij}d(a_j, c_i) \]  
\[ \text{subject to } \sum_{i=1}^{k} w_{ij} = 1 \quad \text{for} \ 1 \leq i \leq k, 1 \leq j \leq N \]  
\[ \sum_{j=1}^{N} y_j = k \]  
\[ w_{ij} \leq y_j \quad \text{for} \ 1 \leq i \leq k, 1 \leq j \leq N \]  
\[ w_{ij}, y_j \in \{0, 1\} \quad \text{for} \ 1 \leq i \leq k, 1 \leq j \leq N \]

where \( y_j \) assumes the value 1 if the element \( a_j \) is selected as one of the centroids and 0 otherwise.

The k-medoids problem is classified as NP-hard and state-of-the-art methods use heuristics to obtain fast near optimal solutions [17]. For this project we implemented the optimal partitioning algorithm suggested by Brusco and Köhn [6] as it offers an efficient way to solve the
optimisation problem (19) using heuristics, while still being able to compute the optimal solution if heuristics have failed. The algorithm consists of three stages illustrated on Figure 2. Since the stage three runs the branch-and-bound algorithm with an embedded Lagrangian relaxation scheme, it guarantees the finding of optimal solution in reasonable time if previous stages did not succeed.

Because this algorithm is not a contribution of this work, we describe the individual stages of the algorithm just briefly and refer the reader to the original paper for more detailed information.

Stage 1: The Vertex Substitution Heuristic

Starting with a random selection of \( k \) elements of \( A \) as the initial set \( C \) compute the sum of distances between all elements and their nearest centroid

\[
E_H := E(\tilde{W}, C) = \sum_{i=1}^{k} \sum_{j=1}^{N} \tilde{w}_{ij} d(c_i, a_j), \tag{24}
\]

with

\[
\tilde{w}_{ij} = \begin{cases} 
1, & \text{if } c_i = \arg \min_{c_i \in C} d(c_i, a_j) \\
0, & \text{otherwise}
\end{cases} \tag{25}
\]

In an iterative process each element in \( A \setminus C \) is evaluated as a substitute for every centroid in \( C \) and the sum of stars is recalculated. At the end of each iteration the substitution with the greatest reduction of sum of stars is made permanent. The iterative process continues until there are no more possible replacements, which yields to a locally optimal solution.

We followed the recommendation in the original paper to restart the algorithm 20 times with different initial sets in order to obtain the upper bound of the globally optimal solution.

Stage 2: Lagrangian Relaxation

Using Lagrangian relaxation on constraint \( \{20\} \) and Lagrangian multipliers \( \lambda \) transforms the problem \( \{19\} \) into the form

\[
\min_{X,Y} E_2(\lambda, W, C) := \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} d(c_i, a_j) + \sum_{i=1}^{N} \lambda_i \left( 1 - \sum_{j=1}^{N} w_{ij} \right), \tag{26}
\]

subject to \( \{21\} \), \( \{22\} \), \( \{23\} \)

Without going into much detail we confine ourselves to noting that to solve the problem we can choose \( k \) elements for which \( \sum_{i=1}^{N} \min(d(c_i, a_j) - \lambda_i, 0) \) is the smallest and then obtain variables \( w_{ij} \) as

\[
w_{ij} = \begin{cases} 
1, & \text{if } y_j = 1 \text{ and } d(c_i, a_j) - \lambda_i < 0 \\
0, & \text{otherwise}
\end{cases} \tag{27}
\]
STAGE 1: Run 20 replications of the VS heuristics to obtain an upper bound on the optimal solution, $E_H$.

STAGE 2: Run the Lagrangian relaxation algorithm with 5 restarts of the procedure with a maximum of 4000 iterations each.

Optimal solution found?

NO

STAGE 3: Run the Branch-and-Bound algorithm with an embedded Lagrangian relaxation scheme.

Return optimal solution

Figure 2: k-medoids clustering by Brusco and Köhn [6].
$E_2(\lambda, W, C)$ is a lower bound on $E$. To find the tightest lower bound $\hat{E}_2$ we solve the Lagrangian dual problem using sub-gradient method. In the case where $\hat{E}_2 = E_H$ the vertex substitution solution is proved to be globally optimal. Brusco and Köhn observed that this is often the case and in particular for small values of $k$.

**Stage 3: Branch-And-Bound Algorithm**

The branch-and-bound algorithm is a widely used technique to solve combinatorial optimisation problems by systematically enumerating all candidate solutions and then traversing through the tree of all candidate solutions and pruning branches with unfeasible solution that do not satisfy lower/upper bounds estimated with some domain specific heuristic [15]. This leads to the reduction of the solution space and--given efficient heuristics--performs better than exhaustive search.

As a heuristic for lower bound estimation the Lagrangian relaxation method from Stage 2 is used with the modification that the centroids fixed in the current branch of the solution space cannot be modified by the algorithm.

**5 Generation of Artificial Trees**

One of the important tasks in the project was the development of algorithms for the generation of trees in controlled way, so that resulting trees will fulfill certain properties. These algorithms were primarily used in the function `RANDOMTREE` in Algorithm 4 to determine centroid trees. Additionally, we used them to generate artificial datasets of trees for our experiments. In the course of the project two algorithms were developed that differ in the assumptions we made about the trees. The first algorithm assumes that the probability for new generations of nodes falls with increasing level while the second algorithm only assumes the constant probability of abortion (first-order Markov property).

For the experiments described in Section 7.1 we used transition probabilities to generate the children of nodes with different labels while the probability that a node will become a leaf was calculated using a discontinuing factor $\gamma \in (0, 1)$ depending on the current level of the tree. Varying this factor, one can influence the size of generated trees although remaining a high degree of randomness.

A transition probability matrix

$$p\text{vals}_{init} = \begin{bmatrix} 0.5 & 0.5 \\ 0.9 & 0.1 \end{bmatrix}$$

(28)

means that a node “1” will get children “1” and “2” with the same probability 0.5, while a node “2” will receive a child “1” with probability 0.9 and a child “2” with probability 0.1.
Algorithm 5 illustrates the generation process in this case. The function RandomTree expects the current node, an alphabet Σ, branching factor, matrix with transition probabilities pvals, vector with probabilities to continue generation process, discontinuity factor γ, and current level of recursion as input parameters. In line 2 the algorithm tests if the generation process should continue. If so, the labels for children nodes are sampled using multinomial distribution from the transition matrix (line 3). For each child the generation process will recursively continue with continuance probability multiplied by γ.

For the experiments with clustering of artificial trees (Sections 7.2 and 7.3) we decided that the generation should satisfy the first-order Markov property, so that the probability for continuing and abortion of the generation process can be summarised in one transition probability matrix.

For example, in order to generate initial centroids for Algorithm 4 with an alphabet Σ = \{1, 2, 3\}, we used following transition probability matrix:

\[
\text{pvals}_{\text{init}} = \begin{bmatrix}
0 & 0.13 & 0.13 & 0.13 \\
0.6 & 0.13 & 0.13 & 0.13 \\
0.6 & 0.13 & 0.13 & 0.13 \\
0.6 & 0.13 & 0.13 & 0.13 \\
\end{bmatrix}
\]

(29)

Figure 3 illustrates the state graph of the generation process. For the node with every label independent of the current level the probability to abort generation process is equal to 0.6 although the probabilities to generate children nodes with any label are equal to 0.13.

Algorithm 6 shows the generation process in this case. The main difference to Algorithm 5 is the absence of the continuance probability and γ. The algorithm looks for abortion probabilities in the first column of pvals instead (line 2). For SampleMultinomial the transition probabilities should be renormalised in order that the sum becomes equal to 1.

## 6 Methods for Evaluation of Results

One of the difficulties we faced evaluating the results from our experiments was the validation of clustering quality. As trees are not embedded in euclidean space, we could not produce 2D graphs often used for this purpose.

The one obvious solution was visual inspection of clusters and centroids, which is limited to a small number of trees and clusters with very distinctive properties.

In the scope of this project we developed a quantitative method for result evaluation. For a given tree we calculate the number of leaves with the same label and the number of parent-child correspondences with the same label for

---

We assume here that the labels are positive integers.

The level of recursion is used to prevent program failure, since with some small probability the generation process can cause a stack overflow.
each possible label from $\Sigma$. This allows to estimate the empirical transition matrix $pvals_{estim}$ for a tree or a forest.

For example, to estimate the transition matrix for the case where $\Sigma = \{1, 2\}$ we calculate following numbers:

- number of “1”-leafs / total number of leaves
- number of “2”-leafs / total number of leaves
- number of “1”-children of “1”-nodes / total number of nodes
- number of “2”-children of “1”-nodes / total number of nodes
- number of “1”-children of “2”-nodes / total number of nodes
- number of “2”-children of “2”-nodes / total number of nodes

The entries of the resulting $pvals$ matrix are then normalised so that the row sum is always equal to 1.

Now we can compare the estimated transition matrix with the true transition matrix used to generate a certain type of trees and calculate the distance between these two matrices as Frobenius norm of their difference.

7 Results

7.1 Evaluation of Different Metrics

The results were achieved after 20 restarts using random search as described in Section 3. The algorithm terminated if no improvement steps were found in the last 500 iterations.
The set of trees of type 1 consists of 30 unordered labelled trees with $\Sigma = \{1, 2\}$, where every node has at most two child nodes. The set of trees is shown in Figure 4. The generation of the trees was performed using

\[
pvals_1 = \begin{bmatrix} 0.5 & 0.5 \\ 0.9 & 0.1 \end{bmatrix}, \quad \text{contProbability}_1 = \begin{bmatrix} 0.95 \\ 0.95 \end{bmatrix}, \quad \text{and} \quad \gamma_1 = 0.85
\]

The resulting trees are presented in Figure 5. One can immediately see that the usage of cTED led to smaller mean and median trees than the usage of MaxSimilarity metrics. Moreover, the usage of cTED led in both cases to the same unordered tree that does not reflect the properties of the dataset. Opposite to cTED, the MaxSimilarity metrics could reproduce the properties of the set much better and the resulting trees are larger.
Figure 5: Results for type 1.
All trees except of median MaxSimilarity 1 tree have subtrees of the form $1 \leftarrow 2 \rightarrow 1$ and no subtrees $2 \leftarrow 2 \rightarrow 2$. The trees median MaxSimilarity 1, mean MaxSimilarity 1, median MaxSimilarity 2, mean MaxSimilarity 4 have subtrees $1 \leftarrow 1 \rightarrow 1$ which represents an undesirable property since the probability for children with both labels from a parent node with label “1” are equal. The trees mean MaxSimilarity 1, median MaxSimilarity 3, mean MaxSimilarity 3, Median MaxSimilarity 4 represent the properties of the data correctly.

Figure 6 illustrates the tree set type 2, also with 30 trees. The set has the following properties:

$$pvals_2 = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}, \text{contProbability}_2 = \begin{bmatrix} 0.995 \\ 0.9 \end{bmatrix}, \text{and } \gamma_2 = 0.85$$

The resulting trees are presented in Figure 7. In this case all metrics were able to reflect the properties of the generator function correctly. Again, the trees from cTED algorithm are smaller compared to the results with MaxSimilarity metrics.

The empirical results are presented in Table 1. For type 1 the trees with Mean MaxSimilarity 2, Mean MaxSimilarity 3 and Median MaxSimilarity 3 show the lowest distance and, as expected, one can make correct conclusion about generation process from these trees. For type 2 the trees from MaxSimilarity 4 median, MaxSimilarity 3 mean and cTED mean show the best quality, however, as mentioned above, the correct conclusions about the generation process can be made from all trees.
Figure 6: Set of trees type 2.
Figure 7: Results for type 2.
In this table as in all following tables we use 0 to denote cTED metric and the numbers between 1 and 4 to denote metrics MaxSimilarity 1 to MaxSimilarity 4 correspondingly.

Table 1: Quantitative results for different metrics.

<table>
<thead>
<tr>
<th>Metric*</th>
<th>Centroid</th>
<th>Distances to pvals</th>
<th>pvals2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>median</td>
<td>0.946 0.769</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>0.946 0.655</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>median</td>
<td>0.933 0.827</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>1.010 0.743</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>median</td>
<td>0.958 0.827</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>0.692 0.883</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>median</td>
<td>0.776 0.691</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>0.743 0.646</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>median</td>
<td>0.823 0.614</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>0.923 0.691</td>
<td></td>
</tr>
</tbody>
</table>

* In this table as in all following tables we use 0 to denote cTED metric and the numbers between 1 and 4 to denote metrics MaxSimilarity 1 to MaxSimilarity 4 correspondingly.

In our experiment MaxSimilarity metrics have shown an overall better performance than cTED. For these reasons and because of the significantly faster calculation of MaxSimilarity metrics, in our clustering experiments we will focus on the usage of MaxSimilarity 2 as not normalised and MaxSimilarity 3 as normalised metrics.

### 7.2 Clustering of Artificial Data: 2 Types

The first experiment with artificial data is very simple. Primarily, it tests the correctness of the algorithms. The dataset contains trees of two types. The trees of the first type contain nodes with labels “1” and “2”, while trees of the second type contain only nodes with label “3”. Transition matrices for generation of the trees are as follows:

\[
\text{pvals}_1 = \begin{bmatrix} 0.4 & 0.3 & 0.3 & 0.0 \\ 0.4 & 0.3 & 0.3 & 0.0 \\ 0.9 & 0.0 & 0.0 & 0.1 \end{bmatrix} \quad (30)
\]

\[
\text{pvals}_2 = \begin{bmatrix} 0.9 & 0.1 & 0.0 & 0.0 \\ 0.9 & 0.0 & 0.1 & 0.0 \\ 0.2 & 0.0 & 0.0 & 0.8 \end{bmatrix} \quad (31)
\]

We generated 30 trees of each kind for clustering and estimated the aggregated transition matrices within the families of trees and the distances...
between these estimated matrices and true matrices used during the generation process

\[ \| \text{pval}^\text{estim}_1 - \text{pval}_1 \| = 0.936 \]  
\[ \| \text{pval}^\text{estim}_2 - \text{pval}_2 \| = 1.294 \]  

These distances show how good can we come to the true generation properties given correct clustering.

Table 2 shows the results of clustering for k-means and k-medians algorithm using MaxSimilarity 2 and MaxSimilarity 3 metrics. These two metrics have shown the best performance for normalised and not normalised metrics in the previous experiment. The distances to \( \text{pvals}_i \) denote the minimal distances between \( \text{pvals}_i \) and estimated transition probability matrices for \( k \) different clusters.

One can see that with exception of k-means with MaxSimilarity 2 we could achieve a perfect differentiation of two families for \( k = 2 \). The k-means algorithm with MaxSimilarity 2 as well as all clustering methods with \( k = 3 \) were very close to the optimal results and indeed could show smaller distances to one or another transition matrices than estimated from the original datasets in (32) and (33).

The algorithm in fact minimises the problem (14). Figure 8 shows the value of the energy function \( E(W, C) \) for different iterations of the k-means algorithm for two examplary runs. In both cases the function monotonically decreases until it becomes constant. At this iteration the algorithm converges.

The advancement in clustering also leads to the decreasing of the distances between transition probability matrices as Figure 9 shows. In the first example on Figure 9(a) two distances between estimated cluster 2 and original cluster 2, and between estimated cluster 3 and original class 1 became slightly smaller while the other distances became larger or remained constant. In the second example (Figure 9(b)) the distances between estimated
Figure 8: Values of the target function $E(W, C)$ for different iterations (clustering of 2 types of artificial data).
cluster 2 and both original classes became smaller, however, it remains clear that the first estimated cluster should be associated with the first original class and the second estimated cluster with the second original class.

The experiments show that the implemented k-means and k-medians algorithm can be in fact used to solve the optimisation problem (14). Furthermore, the distance between the estimated and the original transition probability matrices is a proper measure for quality of this unsupervised learning algorithms.

7.3 Clustering of Artificial Data: 3 Types

The second experiment with artificial data is more sophisticated. The dataset contains 30 trees from three different families generated using following transition matrices:

\[
pvals_1 = \begin{bmatrix}
0.2 & 0.26 & 0.26 & 0.26 \\
0.2 & 0.0 & 0.8 & 0.0 \\
0.2 & 0.0 & 0.0 & 0.8
\end{bmatrix}
\] (34)

\[
pvals_2 = \begin{bmatrix}
0.4 & 0.3 & 0.3 & 0.0 \\
0.4 & 0.1 & 0.1 & 0.4 \\
0.4 & 0.0 & 0.3 & 0.3
\end{bmatrix}
\] (35)

\[
pvals_3 = \begin{bmatrix}
0.2 & 0.4 & 0.0 & 0.4 \\
0.7 & 0.0 & 0.15 & 0.15 \\
0.2 & 0.0 & 0.4 & 0.4
\end{bmatrix}
\] (36)

We performed the clustering of the dataset using k-means, k-medians, and k-medoids algorithms for \( k \in \{2, 3, 4, 5\} \). In this experiment we tested constrained TED as well as all four types of MaxSimilarity metrics. The random search for the centroid was performed using default parameters. Each type of clustering was repeated 20 times and the result with the lowest sum of stars was taken for the further evaluations.

The distances between the aggregated \( pvals_{estim} \) for all trees of certain type and the corresponding \( pvals \) used by generator are as follows:

\[
\|pvals_{estim}^1 - pvals_1\| = 0.397 \quad (37)
\]

\[
\|pvals_{estim}^2 - pvals_2\| = 0.258 \quad (38)
\]

\[
\|pvals_{estim}^3 - pvals_3\| = 0.250 \quad (39)
\]

Table 3 shows the minimal distances to \( pvals \) matrices. Comparably good results were achieved for the MaxSimilarity 1 metric using k-means algorithm with \( k = 4 \) as well as the MaxSimilarity 2 metric using k-means and k-medians algorithms for \( k = 2 \). Constrained TED metric shows better results for \( pvals_2 \) where the distances were commonly smaller, but very high distances for the clusters of the first and third types. While k-means and
Figure 9: Distances to the transition probability matrices $pvals_i$ from generation algorithm for different iteration.
k-medians with MaxSimilarity 3 and MaxSimilarity 4 metrics tend to put all trees into one cluster, and so their performance was especially poor, k-medoids tends to distribute the data more uniformly using normalised metrics than using non-normalised.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Centroid k</th>
<th>Distances to</th>
<th>Cluster sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>pvals₁</td>
<td>pvals₂</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.656</td>
<td>0.351</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.507</td>
<td>0.307</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.676</td>
<td>0.304</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.350</td>
<td>0.349</td>
</tr>
<tr>
<td>0</td>
<td>mean</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.789</td>
<td>0.390</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.872</td>
<td>0.328</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.772</td>
<td>0.308</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.736</td>
<td>0.308</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.710</td>
<td>0.233</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.690</td>
<td>0.233</td>
</tr>
<tr>
<td></td>
<td>medoid</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.748</td>
<td>0.334</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.772</td>
<td>0.308</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.644</td>
<td>0.271</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.467</td>
<td>0.261</td>
</tr>
<tr>
<td>1</td>
<td>mean</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.425</td>
<td>0.290</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.454</td>
<td>0.308</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.495</td>
<td>0.288</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.686</td>
<td>0.347</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.772</td>
<td>0.308</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.690</td>
<td>0.239</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.625</td>
<td>0.239</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.625</td>
<td>0.233</td>
</tr>
<tr>
<td></td>
<td>medoid</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.348</td>
<td>0.314</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.772</td>
<td>0.308</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.595</td>
<td>0.281</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.657</td>
<td>0.292</td>
</tr>
<tr>
<td>2</td>
<td>mean</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.358</td>
<td>0.303</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.654</td>
<td>0.281</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.751</td>
<td>0.290</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.599</td>
<td>0.292</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.772</td>
<td>0.308</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.657</td>
<td>0.308</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.657</td>
<td>0.308</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.830</td>
<td>0.233</td>
</tr>
<tr>
<td></td>
<td>medoid</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The reason for this poor performance lies in the methods we used to select the trial for evaluation. Under conditions of unsupervised learning the minimal sum of stars seems to be the best available indicator, however, it does not always correspond to the best possible clustering. Table 4 shows the trials with the lowest distances to different \( pvals \) matrices among all trials. Obviously, these distances are as good as the best values in Table 3.

This is a very interesting result, however as mentioned above it uses a priori knowledge about the nature of the data that is not available under normal conditions. The methods to select best clustering without prior knowledge is an interesting and important research area, but it does not lie in the scope of the current work.

In this experiment the k-medoids algorithm was proved to be competitive with k-means and k-medians. Its deterministic nature and significantly lower computational time would make the k-medoids algorithm a method of choice for clustering problems where efficiency is the critical property.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Centroid k</th>
<th>Distances to</th>
<th>Cluster sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( pvals_1 )</td>
<td>( pvals_2 )</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td>median</td>
<td></td>
<td>0.881</td>
<td>0.332</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.531</td>
<td>0.303</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.573</td>
<td>0.271</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.361</td>
<td>0.271</td>
</tr>
<tr>
<td>medoid</td>
<td></td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.657</td>
<td>0.333</td>
</tr>
<tr>
<td>mean</td>
<td></td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.798</td>
<td>0.386</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.657</td>
<td>0.345</td>
</tr>
<tr>
<td>median</td>
<td></td>
<td>0.881</td>
<td>0.332</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.575</td>
<td>0.289</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.657</td>
<td>0.271</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.657</td>
<td>0.269</td>
</tr>
</tbody>
</table>

Table 3: Clustering of artificial data: 3 types.


<table>
<thead>
<tr>
<th>Metric</th>
<th>Centroid</th>
<th>k</th>
<th>pvals₁</th>
<th>pvals₂</th>
<th>pvals₃</th>
<th>Cluster sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>mean</td>
<td>4</td>
<td>0.299</td>
<td>0.302</td>
<td>0.599</td>
<td>12, 0, 10, 6</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td>5</td>
<td>0.842</td>
<td>0.338</td>
<td>0.309</td>
<td>0, 0, 7, 17, 4</td>
</tr>
<tr>
<td>4</td>
<td>mean</td>
<td>4</td>
<td>0.870</td>
<td>0.348</td>
<td>0.308</td>
<td>7, 0, 9, 12</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td>4</td>
<td>0.344</td>
<td>0.343</td>
<td>0.516</td>
<td>12, 2, 2, 12</td>
</tr>
</tbody>
</table>

Table 4: Clustering of artificial data: 3 types, best of MaxSimilarity 3 and 4

7.4 Clustering of Real Data

The third experiment is conducted on the real trees that record the observations of biological experiments. The development of cell colonies from single stem cells under different conditions were observed and recorded, carried out in the Lab of Timm Schroeder in the Helmholtz Zentrum München—the German research centre for environmental health. The resulting unordered labelled trees describe the history of the experiments where labels correspond to the types of the cells, and children of a node denote the new cells which the “parent-node” cell has differentiated into. These cell types are: 1–None (no data about the cell type available), 2–Myeloid (mature cell), 4/5–HSC (multipotent progenitors).

The study of the development patterns of hematopoietic (from Ancient Greek “blood generating”) cells, which give the data for this clustering experiment, can help in understanding of molecular mechanisms of the generation of blood cells. That would enable scientists to control the generation process, which is an important step to find a cure for blood disorders like leukaemia.

The dataset used in this experiment was obtained from hematopoietic progenitor cells of mice. The examined cells were Granulocyte-Macrophage-Progenitor (GMP) cells which can rise to granulocytes as well as macrophages. Using some special cytokines (intercellular signalling substances) the development can be led to an offspring set of cells of only one type.

In this section we use the dataset with 198 progenitor cell trees, each of the trees has a GMP cell its root. The dataset contains the trees from three experimental sets:

- 81 trees were cultured with g-csv cytokine, which promotes the generation of granulocytes only,
• 39 trees were treated with m-csf cytokine, which produces only macrophages, and

• 78 trees were given both cytokines (denoted g+m-csf), which resulted in production of both types.

Figure 10 shows the value of the energy function for two exemplary trails. One can see that the algorithm minimises $E(W, C)$ during all 16 iteration steps until it converges.

However, the validation of clustering results is not obvious. The profound evaluation would require an expert knowledge in the domain, so that an attempt to interpret the results is not possible in the scope of this project. Table 6 shows the initial and the last values of the energy function $E(W, C)$, cluster sizes, and distances to the estimated transition probability matrices.

Unlike the results in the previous sections, here we do not have a know generation function with known transition probabilities, and even the assumption of the Markov property of the observed trees is not proven. The matrices $pvals_i$ were estimated from the trees of three different experimental sets described above. The distances between these matrices and the transition probability of the whole dataset, which can serve as a base line, are 0.346, 1.012, and 1.028 correspondingly.

As shown in Table 6, the distances to $pvals_2$ and $pvals_3$ are significantly smaller than the average while the values in the column $pvals_1$ are slightly higher.
Figure 10: Values of the target function $E(W, C)$ for different iterations (clustering of real data).
Algorithm 5 Random Tree Generation 1: with Discontinuity Factor

1: function RANDOMTREE(node, Σ, degree, pvals, contProbability, γ, recursionLevel)
2: if recursionLevel ≤ MAX_RECURSION and Rand() < contProbability[|label| − 1] then
3:   childLabels = SAMPLEMULTINOMIAL(degree, pvals[|label| − 1, 0 : end])
4:   for all childLabel in childLabels do
5:     childNode = NODE(childLabel)
6:     childNode = RANDOMTREE(childNode, Σ, degree, pvals, contProbability · γ, recursionLevel + 1)
7:     ADDCHILD(node, childNode)
8:   end for
9: end if
10: return node
11: end function

Algorithm 6 Random Tree Generation 2: with First-Order Markov Property

1: function RANDOMTREE(node, Σ, degree, pvals, recursionLevel)
2: if recursionLevel ≤ MAX_RECURSION and Rand() > pvals[|label| − 1, 0] then
3:   childLabels = SAMPLEMULTINOMIAL(degree, ∑pvals[|label| − 1,i] = 1
4:   for all childLabel in childLabels do
5:     childNode = NODE(childLabel)
6:     childNode = RANDOMTREE(childNode, Σ, degree, pvals, recursionLevel + 1)
7:     ADDCHILD(node, childNode)
8:   end for
9: return node
10: else
11: return NULL
12: end if
13: end function
### Table 5: Clustering of real data: distances and cluster sizes.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Centroid</th>
<th>$k$</th>
<th>$pvals_1$</th>
<th>$pvals_2$</th>
<th>$pvals_3$</th>
<th>Cluster sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0.553</td>
<td>0.212</td>
<td>0.374</td>
<td>148,33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0.560</td>
<td>0.178</td>
<td>0.264</td>
<td>23,108,50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0.414</td>
<td>0.255</td>
<td>0.061</td>
<td>0,115,24,42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>0.554</td>
<td>0.264</td>
<td>0.203</td>
<td>64,19,74,24</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0.508</td>
<td>0.238</td>
<td>0.425</td>
<td>54,127</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0.515</td>
<td>0.239</td>
<td>0.427</td>
<td>53,0,128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0.508</td>
<td>0.238</td>
<td>0.425</td>
<td>127,0,54,0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>0.550</td>
<td>0.256</td>
<td>0.136</td>
<td>65,32,31,22,31</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0.474</td>
<td>0.423</td>
<td>0.509</td>
<td>107,74</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0.527</td>
<td>0.164</td>
<td>0.238</td>
<td>30,84,67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0.529</td>
<td>0.244</td>
<td>0.342</td>
<td>154,27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>0.547</td>
<td>0.123</td>
<td>0.263</td>
<td>75,78,28</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0.473</td>
<td>0.483</td>
<td>0.560</td>
<td>101,80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0.475</td>
<td>0.438</td>
<td>0.478</td>
<td>77,24,80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0.472</td>
<td>0.368</td>
<td>0.269</td>
<td>32,27,56,66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>0.540</td>
<td>0.187</td>
<td>0.243</td>
<td>67,85,29,0</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 6: Clustering of real data: initial and last energy function values.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Centroid</th>
<th>$E_{start}$</th>
<th>$E_{end}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>2</td>
<td>270086</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>251300</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>264476</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>259664</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td>2</td>
<td>4682</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4957</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>5397</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>4755</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>2</td>
<td>125.762</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>122.770</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>123.331</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>113.737</td>
</tr>
<tr>
<td></td>
<td>median</td>
<td>2</td>
<td>144.482</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>138.547</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>145.346</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>139.988</td>
</tr>
</tbody>
</table>

Table 5: Clustering of real data: distances and cluster sizes.

Table 6: Clustering of real data: initial and last energy function values.
The resulting centroid trees are biologically consistent. Figure 11 illustrates two exemplary centroid trees from the experiment. In these as in most resulting centroid trees the progenitor cell were placed in the root or near the root of the trees, while myeloid cells that can not differentiate further were placed as leafs. Since the generation of initial trees did not use any prior knowledge about cell types, obviously, this architecture was enforced by the algorithm and the dataset.

The experiment shows that the clustering methods developed in the scope of this project can be used not only on artificial data, but also on data from the real world. However, as it is always the case with results from unsupervised learning, the interpretation of the discovered patterns lies in competence of the domain experts.

8 Implementation

The algorithms described in this script were implemented in Python. The Kuhn-Munkres algorithm for solving the assignment problem was implemented as a C-library, which led to a performance increase of about 25% for the calculation of MaxSimilarity metrics.

As the computation of mean trees using random search with several restarts and the calculation of individual distances are very expensive, it was important to parallelise the clustering algorithm.

There already exist some parallelisations of k-means algorithm in the literature [7]. Our problem, however, differs in one significant aspect, and, therefore, we had to develop a new parallel algorithm for k-means. Usually, one uses a parallel version of k-means in order to process large datasets and to minimise the network communication. In contrast, we wanted to
concurrently compute different restarts of the random search algorithm while the dataset remained relatively small.

We decided to implement parallelisation using MapReduce techniques [8], as there are a number of frameworks for different programming languages that support developers by abstracting from the technical details. Since the most of the code was written in Python, we used the PaPy framework for parallelisation.

As described in Section 4.1 k-means alternates between two steps: assignment and update. These steps were mapped as follows on map and reduce operations:

**Assignment**: assign all points to their next centroid and partition them so into clusters

- *assignmentmapper* calculates the distance between each point and each centroid
- *assignmentreducer* for each data point chooses the centroid with minimal distance and assign it to this centroid

**Update**: for each cluster compute the new centroid

- *updatemapper* for each cluster computes a new centroid (random search, most expensive step). Here one can integrate the calculation of several restarts if several “rows” are processed.
- *updatereducer* chooses from many new centroids for the same cluster one with minimal sum of distances.

### Running Times

The computations were performed on the Ubuntu server with AMD Opteron (tm) 8431 processor with 4 cores running in parallel as described above. The running time of the clustering algorithm depends on the number of iterations of the k-means (k-medoids) algorithm needed for convergence. In the small experiment with artificial data described in Section 7.2 the clustering took in average 4 iterations, with in average 74.2 seconds for each iteration. For the more sophisticated experiment with real-world data from Section 7.4 the clustering took in average 9 iterations, with 1198.0 seconds for each iteration.

### 9 Conclusion

This project has shown how the combination of methods from stochastic optimisation, convex optimisation, and machine learning can lead to a solution of the complex problem of tree clustering.

The metric between unordered labelled trees can be obtained using the novel methods like maximal similarity common subtree as well as modification of the established methods like constrained tree edit distance. Our experiments with artificial data has shown the advantage of MaxSimilarity metrics, but cTED could still be interesting for a class of problem that will make use of its constraints. The downside of cTED are the high computational costs compared to MaxSimilarity method.

Another important decision one have to make designing the clustering procedure is the choice of the clustering algorithm. Unlike k-means and k-medians, the k-medoids algorithm does not require random search and so has to be computed only once without loss of quality. The script described an efficient algorithm that always leads to the optimal solution. The stochastic algorithms k-means and k-medians offer a higher flexibility, but have considerably higher computational costs. The project has shown how the elapsed time of computation for these algorithms can be reduced using parallelisation.

The experiments with artificial data has established the validity of algorithms and gave insights in the patterns the clustering algorithm can discover. The results of the application on real-world data, however, can not be fully verified in the scope of this project due to the absence of expert knowledge and understanding of subtle relationships in the data.

As it is often the case, the results of this project reveal new questions. Firstly, the results of clustering of the stem cell data described in Section 7.4 should be evaluated by experts in the domain. Secondly, the methods for centroid search can be further refined. One possible improvement would be the use simulated annealing [13] or variable neighbourhood search [10] as algorithms for stochastic optimisation. The first attempts to use simulated annealing in this project led to significantly increased computational times, but because of the limited scope we could not study the possibilities in more details. Finally, we cannot exclude the possibility for the existence of a sub-gradient that would lead to the desired results. These are the topics for further research.

References


