Practical Course Report

Hierarchical Clustering

Martin Herold

Date of Submission: November 29, 2021
Advisor: Dr. habil. Joachim Spoerhase

Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

1 Introduction

Clustering is a classic problem arising in various fields of computer science and other scientific subjects. The task is to parition the elemnts of an input set into multiple sets. These sets are called clusters. It is desired that the elements inside the same cluster are close to each other. How the term "close"f is exactly defined depends on the concrete problem. But sometimes we want more than dividing our input into clusters. Instead we want a hierarchical structure of the input points. This leads to Hierarchical Clustering. Originally Hierarchical Clustering was no well-defined problem since there was no objective function. It was defined as the structure obtained by agglomerative methods or top-down methods until Dasgupta [Das16] defined an objective function. This results in a new very interesting and complex optimization problem. In this report we analyze this problem and focus on an Euclidean version which allows only one-dimensional data. In Section 3 we look at structural results about our problem. We use this results to compare different algorithms in Section 4. In the end we take a more practical approach and analyze the algorithms on random inputs in Section 5.

2 Definitions

The input of a *Hierarchical Clustering* (HC) is a set of points V with a measure w_{ij} between every pair of distinct points $i, j \in V$. The output is a binary tree $\mathcal{T}$ that represents the optimal Clustering for the points in V. The term optimal is very vague in this context because there are several potential objective functions. Some of them interpret w as a similarity measure while others see it as a dissimilarity measure. We will consider two objective functions, but mainly focus on one. Both base on a similarity measure w and use the term $T(i, j)$ which is the set of leaves below the lowest common ancestor of i and j. The first one is called *Dasgupta's objective* [Das16]

$$\mathcal{F}^{-}(\mathcal{T}) = \sum_{(i,j)\in E} w_{ij}|T(i,j)|.$$

Here for every pair of points from the input we count the number of points that are covered by their lowest common ancestor. Intuitively we want to connect similar points i and j low in the tree. Thus their measure w_{ij} is weighted with a small value since there should not be many points covered by their lowest common ancestor. On the other hand the measure w_{ij} is multiplied by a big number for dissimilar points i and j. Thus connecting similar points low and dissimilar points high results in smaller values of the objective function. Thus the goal is to minimize this objective.

Another objective function is some kind of dual to Dasgupta's objective[CCZY18].

$$\mathcal{F}^{+}(\mathcal{T}) = \sum_{(i,j)\in E} w_{ij}(|V| - |T(i,j)|)$$

Here connecting similar points early and dissimilar points late results in higher values because connecting early means that the multiplicative factor is large. Thus we want to maximize the objective. The HC problem with $\mathcal{F}^-$ is refered to as revenue problem in the literature.

In the definition above, we allowed V to contain arbitrary objects as points. For a variation of HC we restrict V to be a subset of $\mathbb{R}^d$ for a fixed $d \in \mathbb{N}$. Furthermore for $v_i, v_j \in V$ we define $w_{ij} = f(||v_i - v_j||_2)$ for a monotone non-increasing function $f\colon \mathbb{R}_{\geq 0} \to [0,1]$. Then we call w_{ij} a *monotone distance-based similarity measure.* Hierarchical Clustering with a monotone distance-based similarity measure is called *d-dimensional Euclidean Hierarchical Clustering* or more general Euclidean Hierarchical Clustering. A special case we will focus on in this paper is the *1-D HC* which is the 1-dimensional Euclidean Hierarchical Clustering [CCZY18].

A common choice for the monotone non-increasing function f is the *Gaussian kernel.*

$$w_{ij} = \left(\sqrt{2\pi}\sigma\right)^{-n} e^{-\frac{||v_i - v_j||_2^2}{2\sigma^2}}$$

σ is called the normalization factor. It determines the bandwidth of the Gaussian kernel [CCZY18].

Let $\mathcal{T}$ be a HC. Then $V(\mathcal{T})$ denotes the set of labels from the leaves of $\mathcal{T}$.

We define two expressions that help us to deal with sets of points. Let V and W be sets of points.

$$V < W :\Leftrightarrow \forall v \in V \forall w \in W : v < w$$

$$w_{V,W} := \sum_{(v,w) \in V \times W} w_{v,w}$$

3 Structural properties of HC

3.1 Connection between Dasgupta's objective and its dual

Lemma 1. *For every hierarchical clustering $\mathcal{T}$ it holds*

$$\mathcal{F}^-(\mathcal{T}) + \mathcal{F}^+(\mathcal{T}) = n \cdot \sum_{(i,j) \in E} w_{ij}.$$

Proof.

$$\begin{aligned}
\mathcal{F}^-(\mathcal{T}) + \mathcal{F}^+(\mathcal{T}) &= \sum_{(i,j) \in E} w_{ij} |\{x \in T(i,j)\}| + \sum_{(i,j) \in E} w_{ij}(|V| - |\{x \in T(i,j)\}|) \\
&= \sum_{(i,j) \in E} w_{ij}(|\{x \in T(i,j)\}| + (|V| - |\{x \in T(i,j)\}|)) \\
&= \sum_{(i,j) \in E} w_{ij}|V| \\
&= n \cdot \sum_{(i,j) \in E} w_{ij}
\end{aligned}$$

□

It follows that an optimal solution for $\mathcal{F}^-$ is also an optimal solution for $\mathcal{F}^+$. From now on we will focus on $\mathcal{F}^+$.

3.2 Optimal solution has optimal subsolutions

The theorem that is proven here is the foundation for the algorithms in chapter 4.3.

Lemma 2. *Let $\mathcal{T}$ be an optimal HC for V and $\mathcal{T}'$ a rooted subtree of $\mathcal{T}$ with $V(\mathcal{T}') = V'$. Then $\mathcal{T}'$ is an optimal solution for V'.*

Proof. Assume there is a HC $\mathcal{T}^*$ of V' with $\mathcal{F}^+(\mathcal{T}') < \mathcal{F}^+(\mathcal{T}^*)$. Let $\hat{\mathcal{T}}$ be the HC which results from $\mathcal{T}$ by replacing $\mathcal{T}'$ with $\mathcal{T}^*$. We define $\overline{T}(i,j) = |V| - |\{x \in T(i,j)\}|$.

$$\begin{aligned}
\mathcal{F}^+(\mathcal{T}) &= \sum_{(i,j)\in E} w_{ij}(|V| - |\{x \in T(i,j)\}|) \\
&= \sum_{\{i,j\}\in\binom{V\setminus V'}{2}} w_{ij}\overline{T}(i,j) + \sum_{\{i,j\}\in\binom{V'}{2}} w_{ij}\overline{T}(i,j) + \sum_{(i,j)\in(V\setminus V')\times V'} w_{ij}\overline{T}(i,j) \\
&= \sum_{\{i,j\}\in\binom{V\setminus V'}{2}} w_{ij}\overline{T}(i,j) + \sum_{\{i,j\}\in\binom{V'}{2}} w_{ij}(\overline{T}'(i,j) + |V \setminus V'|) + \sum_{(i,j)\in(V\setminus V')\times V'} w_{ij}\overline{T}(i,j) \\
&= \sum_{\{i,j\}\in\binom{V\setminus V'}{2}} w_{ij}\overline{T}(i,j) + \sum_{\{i,j\}\in\binom{V'}{2}} w_{ij}\overline{T}'(i,j) + \sum_{\{i,j\}\in\binom{V'}{2}} w_{ij}|V \setminus V'| + \sum_{(i,j)\in(V\setminus V')\times V'} w_{ij}\overline{T}(i,j) \\
&= \sum_{\{i,j\}\in\binom{V\setminus V'}{2}} w_{ij}\overline{T}(i,j) + \mathcal{F}^+(\mathcal{T}') + \sum_{\{i,j\}\in\binom{V'}{2}} w_{ij}|V \setminus V'| + \sum_{(i,j)\in(V\setminus V')\times V'} w_{ij}\overline{T}(i,j) \\
&< \sum_{\{i,j\}\in\binom{V\setminus V'}{2}} w_{ij}\overline{T}(i,j) + \mathcal{F}^+(\mathcal{T}^*) + \sum_{\{i,j\}\in\binom{V'}{2}} w_{ij}|V \setminus V'| + \sum_{(i,j)\in(V\setminus V')\times V'} w_{ij}\overline{T}(i,j) \\
&= \sum_{\{i,j\}\in\binom{V\setminus V'}{2}} w_{ij}\overline{T}(i,j) + \sum_{\{i,j\}\in\binom{V'}{2}} w_{ij}\overline{T}^*(i,j) + \sum_{\{i,j\}\in\binom{V'}{2}} w_{ij}|V \setminus V'| + \sum_{(i,j)\in(V\setminus V')\times V'} w_{ij}\overline{T}(i,j) \\
&= \mathcal{F}^+(\hat{\mathcal{T}})
\end{aligned}$$

Thus $\hat{\mathcal{T}}$ is a better solution than $\mathcal{T}$. This contradicts the optimality of $\mathcal{T}$. Therefore the assumption is wrong and a better solution $\mathcal{T}^*$ can not exist. $\square$

3.3 Optimal 1D-HC is laminar

A *laminar* solution is a tree where every two distinct subtrees do not intersect geometrically. Another equivalent description is that every subtree T covers an intervall of the input. I.e. for an input $V = \{v_1, v_2, \ldots, v_n\}$ with $v_i < v_{i+1}$ we have $V(T) = \{v_k | k \in [i,j]\}$.

In this section we will focus on the hypothesis whether the optimal 1D-HC is laminar. As we will see later in chapter 4.3 this question has massive impact on the difficulty of calculating the optimal HC. There are a few indications that this holds but it appears more difficult to show than expected and remains unproven in this report.

At first we will see a proof restricted to four vertices. We can derive an algorithm that checks for any n if every solution with at most n vertices is laminar. For $n \in \{5, 6\}$ this holds but it remains unclear if the algorithm will return "true" for larger n.

Theorem 3. *For every set V with $|V| \leq 4$ there is a laminar optimal solution.*

Proof. We distinguish between two cases. The first one is that there is no non-laminar solution for V. Then there is a laminar optimal solution. In the other case we have a non-laminar optimal solution $\mathcal{T}$.

We name the vertices $V = \{A, B, C, D\}$ with $A < B < C < D$. Since there are only finite HCs for the finite input set V, we can look at all of possible cases. They are shown in Figure 1.

- $\mathcal{T}_1 = \{\{\{A\}, \{C\}\}, \{\{B\}, \{D\}\}\}$

  We apply Dasgupta's objective.

  $$\mathcal{F}^+(\mathcal{T}_1) = 2 \cdot w_{AC} + 2 \cdot w_{BD}$$

  Since closer points are always more similar, we see $w_{AB} > w_{AC}$ and $w_{CD} > w_{BD}$. This can be used to see:

  $$\mathcal{F}^+(\{\{\{A\}, \{B\}\}, \{\{C\}, \{D\}\}\}) = 2 \cdot w_{AB} + 2 \cdot w_{CD} > \mathcal{F}^+(\mathcal{T}_1)$$

- $\mathcal{T}_2 = \{\{\{A\}, \{D\}\}, \{\{B\}, \{C\}\}\}$

  We apply Dasgupta's objective again.

  $$\mathcal{F}^+(\mathcal{T}_2) = 2 \cdot w_{AD} + 2 \cdot w_{BC}$$

  We see $w_{AB} > w_{AC} > w_{AD}$ and it follows:

  $$\mathcal{F}^+(\{\{\{A\}, \{\{B\}, \{C\}\}\}, \{D\}\}) = 2 \cdot w_{BC} + w_{AB} + w_{AC} > \mathcal{F}^+(\mathcal{T}_2)$$

- $\mathcal{T}_3 = \{\{\{\{A\}, \{C\}\}, \{B\}\}, \{D\}\}$

  We apply Dasgupta's objective again.

  $$\mathcal{F}^+(\mathcal{T}_3) = 2 \cdot w_{AC} + w_{AB} + w_{BC}$$

  We see $w_{BC} > w_{AC}$ and it follows:

  $$\mathcal{F}^+(\{\{\{A\}, \{\{B\}, \{C\}\}\}, \{D\}\}) = 2 \cdot w_{BC} + w_{AB} + w_{AC} > \mathcal{F}^+(\mathcal{T}_3)$$

These three cases are complete with respect to symmetry. Therefore every set V with cardinality four has an optimal laminar HC. $\square$

This statement also holds for five points and can be proven with the same proof idea. Since we only have to find a better HC for every non-laminar HC where "better" can be proven by the simple argument of monotony. Thus question "is there an optimal solution to every set V with $|V| \leq n$ which is laminar" can be answered for any $n \in \mathbb{N}$ by an algorithm.

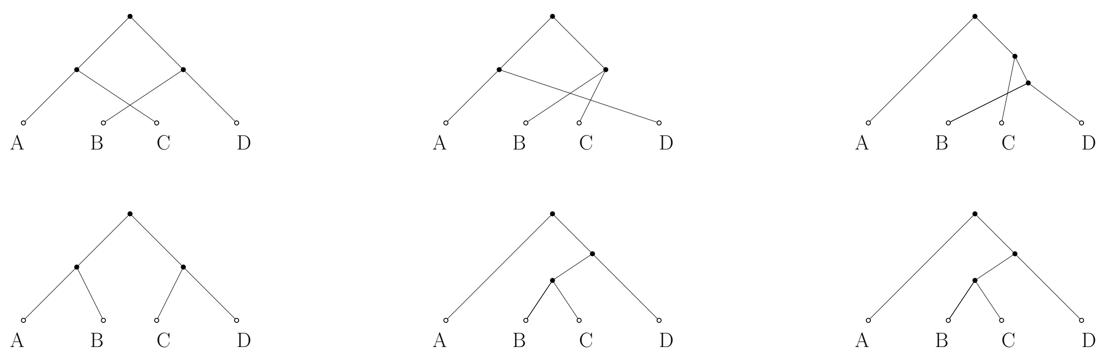

Fig. 1: The three cases of non-laminar solutions and the better solutions from the proof

4 Algorithms

4.1 Average Linkage

Average Linkage is a classic algorithm. It was originally designed for the Clustering problem without hierarchical structure. We start with n initial clusters where the input points are evenly distributed on. The pair of closest clusters are merged repeatedly until a certain condition holds. For example until only a certain number of clusters is left. The term "close" can be interpreted in different ways and will be discussed later on. These remaining clusters are the clustering returned by the algorithm.

If not only the remaining clusters are considered but also how they and in which order they are assembled from the initial clusters, this results in an algorithm for HC. Every merging step of two clusters is represented as a node whose children are these two clusters.

In fact the definition above describes agglomerative clustering methods in general and not only Average-Linkage. The differences between the different agglomerative clustering methods are the interpretation of the word "closest". Single-Linkage looks for example at the closest pair of points from different clusters and combines the clusters that contain these two points. Average-Linkage considers every data-point of a cluster to determine its (dis)similarity to another cluster. The (dis)similarity between two clusters A and B is considered as the average over all pairs of points from the two clusters:

$$w(A, B) = \frac{\sum\limits_{a \in A, b \in B} w_{ab}}{|A| \cdot |B|}$$

If w is a similarity measure, this value needs to be maximized. Otherwise, it should be minimized. Average-Linkage is a popular representative of this group because it not only performs well in practice but also several theoretical guarantees are known for this algorithm. As all agglomerative clustering methods it is easy to implement. The algorithm can be seen here 1. The term $\mathcal{S}$ denotes the only element in $\mathcal{S}$.

Algorithm 1: AverageLinkage(V)

```
   Input: set of points $V$
   Output: Hierarchical Cluster $\mathcal{T}$
1  $\mathcal{S} = \{\{v_1\}, \{v_2\}, ..., \{v_n\}\}$
2  while $|\mathcal{S}| > 1$ do
3      $S_1, S_2 = \min_{S_1, S_2 \in \mathcal{S} \wedge S_1 \neq S_2} w(S_1, S_2)$
4      $\mathcal{S} = \{\{S_1, S_2\}\} \cup \mathcal{S} \setminus \{S_1, S_2\}$
5  return $\mathcal{S}[0]$
```

4.1.1 Runtime

A trivial upper bound for the runtime is $O(n^3)$ since we have linearly many merging steps and for each step we need to compute the average distance between each pair of clusters. Since every pair of data points is considered at most once per step, this computation takes at most quadratic time. This results in total in a cubic runtime. We will see that this bound is not tight.

Manning et. al.[MRS08] showed that Average linkage can be implemented in $O(|V|^2 \log |V|)$ time.

The special case of 1D-HC satisfies a special property that can be used to improve the runtime.

Lemma 4. *For three sets of points V_1, V_2, V_3 with*

$$V_1 < V_2 < V_3$$

it holds

$$w(V_1, V_2) \leq w(V_1, V_3)$$

if w is a dissimmilarity measure (for simmilarity vice versa).

Proof. Let v_2^* be the largest point of V_2 and v_3^* the smallest point of V_3.

$$\begin{aligned}
w(V_1, V_2) &= \frac{1}{|V_1| \cdot |V_2|} \sum_{(v_1,v_2) \in V_1 \times V_2} w_{v_1,v_2} \\
&\leq \frac{1}{|V_1| \cdot |V_2|} \sum_{(v_1,v_2) \in V_1 \times V_2} w_{v_1,v_2^*} \\
&= \frac{1}{|V_1| \cdot |V_2|} \sum_{v_1 \in V_1} w_{v_1,v_2^*} \cdot |V_2| \\
&= \frac{1}{|V_1|} \sum_{v_1 \in V_1} w_{v_1,v_2^*} \\
&\leq \frac{1}{|V_1|} \sum_{v_1 \in V_1} w_{v_1,v_3^*} \\
&= \frac{1}{|V_1| \cdot |V_3|} \sum_{v_1 \in V_1} w_{v_1,v_3^*} \cdot |V_3| \\
&\leq \frac{1}{|V_1| \cdot |V_3|} \sum_{(v_1,v_3) \in V_1 \times V_3} w_{v_1,v_3} \\
&= w(V_1, V_3)
\end{aligned}$$

□

Lemma 5. *Let V_1, V_2, W be set of points. Then it holds*

$$w(V_1 \cup V_2, W) = \frac{|V_1| \cdot w(V_1, W) + |V_2| \cdot w(V_2, W)}{|V_1| + |V_2|}$$

Proof.

$$\begin{aligned}
w(V_1 \cup V_2, W) &= \frac{1}{|V_1 \cup V_2| \cdot |W|} \sum_{(v,w)\in(V_1\cup V_2)\times W} w_{v,w} \\
&= \frac{1}{(|V_1| + |V_2|) \cdot |W|} \sum_{(v,w)\in V_1\times W} w_{v,w} + \sum_{(v,w)\in V_2\times W} w_{v,w} \\
&= \frac{1}{|V_1| + |V_2|} \left(\frac{\sum\limits_{(v,w)\in V_1\times W} w_{v,w}}{|W|} + \frac{\sum\limits_{(v,w)\in V_2\times W} w_{v,w}}{|W|} \right) \\
&= \frac{1}{|V_1| + |V_2|} \left(|V_1| \cdot \frac{\sum\limits_{(v,w)\in V_1\times W} w_{v,w}}{|W| \cdot |V_1|} + |V_2| \cdot \frac{\sum\limits_{(v,w)\in V_2\times W} w_{v,w}}{|W| \cdot |V_2|} \right) \\
&= \frac{1}{|V_1| + |V_2|} \left(|V_1| \cdot w(V_1, W) + |V_2| \cdot w(V_2, W)\right) \\
&= \frac{|V_1| \cdot w(V_1, W) + |V_2| \cdot w(V_2, W)}{|V_1| + |V_2|}
\end{aligned}$$

□

Theorem 6. *Average-Linkage can be implemented in $O(|V|^2)$ time for the one dimensional case.*

Proof. From Lemma 4 it follows that Average-Linkage only has to consider pairs of neighbouring clusters for the merging step.

So the algorithm needs to keep track of linear many distances between neighbouring clusters and find the minimum value. This can be done in linear time per iteration. Since there are $O(|V|)$ iterations, this results in qudratic time in total. It remains to show that these distances can be efficiently updated throughout the merging steps.

In every iteration the distance between the newly assembled cluster an its both neighbours has to be computed. This may take quadratic time since the new cluster and its neighbours can have linear many points. This estimation results in a cubic total runtime which is not sufficient to proove the theorem. Thus this estimation needs to be improved.

Let L be the left neighbour of the new cluster that was merged from V_1 and V_2 such that $V_1 < V_2$. Since V_1 and L are neighbourghed we already know $w_{V_1,L}$. Using Lemma 5 only $w_{V_2,L}$ has to be computed manually. So we only sum over pairs of points that are now in neighbouring clusters but weren't before in the updating step. Since a pair of points can have this property only in one iteration because afterwards they are in neighbouring clusters, each pair is considered only once in the updating of the distances between the clusters. This takes in total only quadratic time. □

4.1.2 Correctness

We will see that AL is a good approximation for random inputs particularly if you consider its runtime. But we will also prove theoretical results about the output of Average Linkage.

Theorem 7 ([CCZY18])**.** *Average Linkage is a 1/2-approximation for 1D-HC.*

Notice that this result holds for the general Hierarchical Clustering problem. Since the 1D-HC is special case we can obtain the following corollary.

Corollary 8. *Average Linkage is a 1/2-approximation for Euclidean-1D-HC.*

Theorem 9. *Average Linkage is not optimal for 1D-HC with a Gaussian kernel.*

Proof. W.l.o.g. we assume $\sigma = 1$. We will look at $V = \{0, 1.01, 2, 3\}$.

Here Average Linkage would return

$$AL(V) = \{\{0\}, \{\{\{1.01\}, \{2\}\}, \{3\}\}\}.$$

We apply Dasguptas objective

$$\begin{aligned}\mathcal{F}^+(AL(V)) &= (\sqrt{2\pi})^{-4} \cdot (2 \cdot e^{-\frac{2-1.01}{2}} + e^{-\frac{3-2}{2}} + e^{-\frac{3-1.01}{2}}) \\ &\approx 0.0688\end{aligned}$$

But there is a solution

$$O = \{\{\{0\}, \{1.01\}\}, \{\{2\}, \{3\}\}\}$$

with

$$\mathcal{F}^+(O) \approx 0.0787.$$

So AL doesn't return the optimal solution for V and therefore it isn't optimal.

□

4.2 Sparsest Cut

In contrast to Average-Linkage which is a bottom-up approach, we will now consider a top-down approach. That means we start with the complete input set V and divide it recursively in two non-empty and disjunct subsets.

It should be divided into V_1, V_2 such that

$$\sum_{(v_1,v_2)\in V_1\times V_2} \frac{w_{v_1,v_2}}{|V_1| \cdot |V_2|}$$

is minimized.

This problem is called the Sparsest Cut problem and it is NP-hard. Thus unless we proof P = NP, we have to focus on approximation algorithms.

4.2.1 Runtime

Since we didn't fix any algorithm for solving the Sparsest Cut problem we can not prove anything concrete about the runtime. But we can observe that the runtime is upperbounded by

$$O(n \cdot f(n))$$

where f is the runtime for solving the Sparsest Cut problem. This holds since we can have at most linear recursive calls. Because a set can be divided at most in linear many non-empty and disjunct subsets.

4.2.2 Correctness

As statet above we will not consider correct algorithms but approximation algorithms.

Theorem 10 ([Das16])**.** *An α-approximation for the Sparsest Cut problem yields an $(\alpha \log n)$-approximation for HC with Dasgupta's objective(using algorithm 2)*

But this technique has its limit since we can prove it is not optimal even if Sparsest Cut can be solved optimally.

Theorem 11. *If SC is an algorithm that solves the Sparsest Cut problem optimal, then algorithm 2 is not optimal for 1D-HC with a gaussian kernel.*

Proof. W.l.o.g. we assume $\sigma = 1$. Let $V = \{0, 1.1, 2, 3\}$.

Sparsest Cut would give the following solution

$$SC(V) = \{\{0\}, \{\{\{1.1\}, \{2\}\}, \{3\}\}\}.$$

We apply Dasguptas objective.

$$\begin{aligned}
\mathcal{F}^+(SC(V)) &= (\sqrt{2\pi})^{-4} \cdot (e^{-\frac{(2-1.1)^2}{2}} + e^{-\frac{(3-1.1)^2}{2}} + e^{-\frac{(3-2)^2}{2}}) \\
&\approx 0.0714 \\
&< 0.1025 \\
&\approx (2\sqrt{2\pi})^{-4} \cdot (e^{-\frac{(1.1-0)^2}{2}} + e^{-\frac{(3-2)^2}{2}}) \\
&= \mathcal{F}^+(\{\{\{0\}, \{1.1\}\}, \{\{2\}, \{3\}\}\})
\end{aligned}$$

□

4.3 Optimal

In this section we will look at two similar algorithms. One is guaranteed to return the costs of an optimal 1D-HC while taking exponential time and the other returns the costs of an optimal laminar 1D-HC in polynomial time. If the hypothesis that every 1D-HC instance has an optimal solution that is laminar holds, this means that an optimal 1D-HC can be computed effinciently using the second argument.

Algorithm 2: SparsestCut(V)

```
Input: set of points $V$
Output: Hierarchical Cluster $\mathcal{T}$
1 if $|V| \leq 1$ then
2     return $\{V[0]\}$
3 $V_1, V_2 = SC(V)$
4 return $\{SparsestCut(V_1), SparsestCut(V_2)\}$
```

Algorithm 3: Optimal(V)

```
Input: set of points $V$
Output: minimal value of $\mathcal{F}^+(\mathcal{T})$
1 for $V' \subseteq V$ do
2     $O[V'] = 0$
3 for $v \in V$ do
4     $O[\{v\}] = 0$
5 for $i = 2$ to $n$ do
6     for $V' \in \binom{V}{i}$ do
7         for $L \subseteq V' \wedge L \neq \emptyset \wedge L \neq V'$ do
8             $O[V'] = \max\{O[V'], O[L] + O[V' \setminus L] + i \cdot w_{L,R}\}$
9 return $O[V]$
```

4.3.1 Correctness

At first we will show that our two algorithms fullfill the requirements above.

Theorem 12. *The algorithm 3 optimizes the objective $\mathcal{F}^+$.*

Proof. We will proof this by induction over the cardinality of V. We call it n.

- (IB) $n = 1$:

  This case is easy since there is only one HC for a set with one item.

- (IS) $(n \to n + 1)$:

  We assume that the statement holds for $|V| \leq n$ and show that it follows for $|V| = n + 1$.

  Let $\mathcal{T}^* = \{\mathcal{T}_1, \mathcal{T}_2\}$ be the optimal HC with $V(\mathcal{T}^*) = V$. From theorem 2 it follows that $\mathcal{T}_1$ is the optimal HC that covers $V(\mathcal{T}_1)$. The analog statement holds for $\mathcal{T}_2$. Since the algorithms iterates over every non-empty, proper subset of V, it has $L = V(\mathcal{T}_1)$ in one iteration. The term in the eigth line has the value $\mathcal{F}^+(\hat{\mathcal{T}})$ for $\hat{\mathcal{T}} = \{\hat{\mathcal{T}}_1, \hat{\mathcal{T}}_2\}$ where $\hat{\mathcal{T}}_1$ (resp. $\hat{\mathcal{T}}_2$) is the optimal HC covering L (resp. $V \setminus L$). By theorem 2 we see $\mathcal{T}_1 = \hat{\mathcal{T}}_1$ and $\mathcal{T}_2 = \hat{\mathcal{T}}_2$. Thus it holds after iterating over all sets L that $O[V] = \mathcal{F}^+(\mathcal{T}^*)$ and the theorem is shown.

Algorithm 4: Optimal-Laminar(V)

```
Input: set of points $V = \{v_1, v_2, \dots, v_n\}$
Output: minimal value of $\mathcal{F}^+(\mathcal{T})$
1  for laminar $\mathcal{T}$s for intervalls $V' \subseteq V$ do
2      $O[V'] = 0$
3  for $v \in V$ do
4      $O[\{v\}] = 0$
5  for $i = 2$ to $n$ do
6      for $j = 1$ to $n - i + 1$ do
7          for $k = 1$ to $i - 1$ do
8              $V' = \{v_j, v_{j+1}, \dots, v_{j+(i-1)}\}$
9              $L = \{v_j, v_{j+1}, \dots, v_{j+(k-1)}\}$
10             $O[V'] = \max\{O[V'], O[L] + O[V' \setminus L] + i \cdot w_{L,R}\}$
11 return $O[V]$
```

□

Theorem 13. *The algorithm 4 optimizes the objective $\mathcal{F}^+$ if only laminar solutions are allowed.*

Proof. This theorem can be proven almost the same way as theorem 12. The only difference beeing that subtrees of a laminar solution have to cover right/left intervals of the input set. □

4.3.2 Runtime

After showing the correctness we will now consider the runtime of our two algorithms.

Theorem 14. *Optimal(V) runs in $O^*(3^n)$ with $|V| = n$.*

Proof. Since the computations of the eighth line can be done in polynomial time, we can assume it to take constant time. This is because the O^* notation neglects polynomial terms.

Thus it has to be considered how many times the eighth line is executed. A first approach is to see that the two For-Loops iterate over subsets of V which results in an upper bound for the runtime of

$$2^n \cdot 2^n = 4^n$$

But this analysis is not tight.

Since the loop over L iterates only over subsets of V' which has cardinality i. This results in the following upper bound.

$$\begin{aligned}\sum_{i=1}^{n} \binom{n}{i} \cdot 2^i &= \sum_{i=1}^{n} \binom{n}{i} \cdot 2^i \cdot 1^{n-i} \\ &= (2+1)^n \\ &= 3^n\end{aligned}$$

□

Theorem 15. *Optimal-Laminar(V) runs in $O(n^4)$ with $|V| = n$.*

Proof. Since there are three For-Loops iterating over a range $\leq n$, the tenth line is executed $O(n^3)$ times. If $w_{L,R}$ is computed naively $(O(n^2))$ this would result in the runtime $O(n^5)$. But this can be improved.

We look closer at $w_{L,R}$ for easy/similar inputs.

$$w_{\{v\},\{u_1,u_2,\ldots,u_m\}} = \sum_{i=1}^{m} w_{v,u_i}$$

This means the tenth line can be computed in linear time if $k = 1$. There is also a result for $k > 1$.

$$\begin{aligned}w_{\{v_1,v_2,\ldots,v_m,s\},\{u_1,u_2,\ldots,u_l\}} &= \sum_{i=1}^{l} \left(w_{s,u_i} + \sum_{j=1}^{m} w_{v_j,u_i} \right) \\ &= \sum_{i=1}^{l} w_{s,u_i} + \sum_{i=1}^{l} \sum_{j=1}^{m} w_{v_j,u_i} \\ &= \sum_{i=1}^{l} w_{s,u_i} + \sum_{i=1}^{l} \sum_{j=1}^{m} w_{v_j,u_i} + \sum_{j=1}^{m} w_{v_j,s} - \sum_{j=1}^{m} w_{v_j,s} \\ &= \sum_{j=1}^{m} \sum_{i=1}^{l} w_{v_j,u_i} + \sum_{j=1}^{m} w_{v_j,s} + \sum_{i=1}^{l} w_{s,u_i} - \sum_{j=1}^{m} w_{v_j,s} \\ &= \sum_{j=1}^{m} \left(w_{v_j,s} + \sum_{i=1}^{l} w_{v_j,u_i} \right) + \sum_{i=1}^{l} w_{s,u_i} - \sum_{j=1}^{m} w_{v_j,s} \\ &= w_{\{v_1,v_2,\ldots,v_m\},\{u_1,u_2,\ldots,u_l,s\}} + \sum_{i=1}^{l} w_{s,u_i} - \sum_{j=1}^{m} w_{v_j,s}\end{aligned}$$

Thus the tenth line can also be computed in linear time if $k > 1$ when the value for $w_{L,R}$ from the last iteration is remembered.

This results in a total running time of $O(n^3)$. □

n	m	AL	SC	COMBINED	$Pr(AL)$	$Pr(SC)$	$Pr(COMBINED)$
4	100,000	1,154	8,840	1,069	1.15%	8.84%	1.07%
6	10,000	574	3,367	506	5.74%	33.67%	5.06%
8	10,000	1,386	5,835	1.243	13.86%	58.35%	12.43%
10	2,000	449	1,565	407	22.45%	78.25%	20.35%
12	200	53	169	49	26.5%	84.5%	24.5%

Tab. 1

5 Experiments

In this section we will see how good the algorithms perform on random inputs of different size with focus on the probability of being optimal.

5.1 Set up

At first we will describe the set up that was used for running the experiments.

5.1.1 Hardware

The experiments were run on a laptop. The processor is an Intel Core i7-4720HQ with eight cores. The RAM has a size of 8 GB.

5.1.2 Software

The laptop is running Ubuntu 18.04.5 LTS (64-bit). The algorithms were implemented in C++. They were compiled using g++ 7.5.0.

5.1.3 Input and goal function

The inputs were chosen independently and uniformly at random from the interval $[0, 3]$. For the Gaussian kernel the standard derivation $\sigma = 1$ was chosen.

5.2 Results

Table 1 shows the amount of times that Average Linkage (AL) (resp. Sparsest Cut (SC) resp. both algorithms (COMBINED)) were worse than the optimum. n represents the cardinality of the input set and m represents the number of input sets that were tested for a specific n. We define AL (resp. SC) as the number of instances where Average Linkage (resp. Sparsest Cut) did not return the optimal value. $COMBINED$ is the number of instances where neither of the algorithms computes the optimal value. $Pr(X)$ is the relative proportion of instances at which event X occured ($^X/_m$) for $X \in \{AL, SC, COMBINED\}$.

Interestingly all probabilities increase for increasing n. This may be explained by the fact that a larger input is more likely to have complex structes than a smaller one.

n	m	$Pr(COMBINED|AL)$
4	100,000	92.63%
6	10,000	88.15%
8	10,000	89.68%
10	2,000	90.65%
12	200	92.45%

Tab. 2

Furthermore it holds for all n that $Pr(AL)$ is significant smaller than $Pr(SC)$. It means that Average Linkage is more likely to perform optimal than Sparsest Cut. Another noticeable fact is that almost every input where Average Linkage isn't optimal also Sparcest Cut isn't optimal. Table 2 concentrates on this relationship by showing the value ${}^{Pr(COMBINED)}/_{Pr(AL)}$.

Unexpectedly this value is the lowest for $n = 4$. Then goes to a minimum at $n = 6$ and starts monotonically growing. The behaviour starting at $n = 6$ can be explained because if a set contains four points which can't be handled optimal by Average Linkage but by Sparsest Cut, there are more points left to be structured in a way that Sparsest Cut doesn't find an optimal solution. Such a structure is more likely formed if there are more points. Whereas we don't have an explanation for the maximum at $n = 4$.

References

[CCZY18] Moses Charikar, Vaggos Chatziafratis, Rad Ziazadeh, and Grigory Yaroslavtsev: Hierarchical clustering for euclidean data. *arXiv Computer Science*, 2018.

[Das16] Sanjoy Dasgupta: A cost function for similarity-based hierarchical clustering. *STOC '16: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, 2016.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze: *Introduction to Information Retrieval*, chapter 17 Hierarchical Clustering, pages 389–390. https://nlp.stanford.edu, 2008.