FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Approximation Algorithms for Clustering
Student:	Vladimir Ananyev
Supervisor:	Ing. Tomáš Borovička
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2019/20

Instructions

Cluster analysis is an unsupervised method for identifying groups of similar instances in a dataset. However, cluster analysis becomes a challenging task for larger datasets when the computational complexity of the algorithm makes the computation of exact solution infeasible. However, complexity can be reduced with sufficient results by utilizing approximation algorithms.

1) Review and theoretically describe state of the art algorithms for approximation of clustering.
2) Implement at least one of the reviewed algorithms for clustering approximation.
3) Experimentally compare the exact and approximate method in terms of time complexity and clustering quality on various datasets.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 13, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Approximation Algorithms for Clustering

Vladimir Ananyev

Department of Theoretical Computer Science
Supervisor: Ing. Tomáš Borovička

January 10, 2019

Acknowledgements

I would like to thank my supervisor for providing great feedback on my thesis and for pushing me to do better. I would like to thank my parents and Karina Kulaga for their support and encouragement. I would also like to thank Hoang Huy Vu, Markéta Jůzlová, and Valeria Iegorova for their help and constructive criticizm.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on January 10, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Ananyev, Vladimir. *Approximation Algorithms for Clustering.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Aglomerativní hierarchické shlukování je důležitý shlukovací algoritmus, který má mnoho praktických využití, na příklad pro segmentaci trhu. Jeho největší nevýhodou je jeho velká časová složitost $\mathcal{O}(n^3)$. Cílem této práce je popsat a zanalyzovat algoritmy aproximující aglomerativní hierarchické shlukování. Tyto algoritmy mají nižší časovou složitost a produkují srovnatelné výsledky s exaktními metodami. Experimenty ukázaly, že aproximační algoritmus LSH-link je signifikantně rychlejší na velkých datech než exaktní algoritmus MST-linkage algoritmus.

Klíčová slova Hierarchické shlukování, LSH-link, MST-linkage, hybridní shlukování

Abstract

Agglomerative hierarchical clustering is an important clustering algorithm which has many real life applications such as customer segmentation. Its biggest drawback is its large time complexity of $\mathcal{O}(n^3)$. This thesis presents and describes approximation algorithms to the agglomerative hierarchical clustering. Such algorithms have lower time complexity and produce similar results to the agglomerative hierarchical clustering. The experiments showed that for the large data sets the approximation method LSH-link performed significantly faster then the MST-linkage, an agglomerative hierarchical clustering algorithm for the single linkage.

Keywords Hierarchical clustering, LSH-link, MST-linkage, Hybrid clustering

Contents

List of Figures

List of Tables

Introduction

As data becomes more and more available and the amount of data grows exponentially, there is a need to extract valuable information from that data. Today, many different approaches exist in order to achieve that goal. One of such approaches is called clustering. Clustering is used to structure the data in order to discover different possible groupings within the data. It has many practical applications in the areas such as medicine, information science and biology [1]. One common clustering method is called hierarchical clustering. Hierarchical clustering partitions the data in a way that it is possible to observe the obtained groupings at different levels of granuality. This property makes hierarchical clustering an important tool for observing complex patterns. However, due to its high computational time complexity, its usage is inpractical with large data sets, which are common in many problems. In order to overcome this limitation, there exist several approximation methods that attempt to produce similar results to the standard algorithm while reducing the running time of the algorithm. In this thesis, some of those approximation methods will be presented and described in detail. Some of those algorithms will be compared to the exact algorithm in terms of clustering performance, and the required computation time complexity. The theoretical background required to understand the hierarchical clustering algorithm will be presented in the following chapters.

Chapter 1

The clustering problem

In real world there are many occasions when it is necessary to perform groupings. For example, clustering can be used to group the human genes, or it can be used to detect several target groups among the users of some service. Regardless of the task, the goal of clustering is to produce a grouping of objects, such that the objects within each group are as similar as possible, and the objects from different groups are as different as possible. However, it is difficult to say if such grouping is correct for a specific task. Even if the obtained grouping satisfies the general definition of clustering, it might not be suitable for a specific domain. Therefore, it is very difficult or even impossible to justify a specific clustering result without the domain knowledge. In Figure

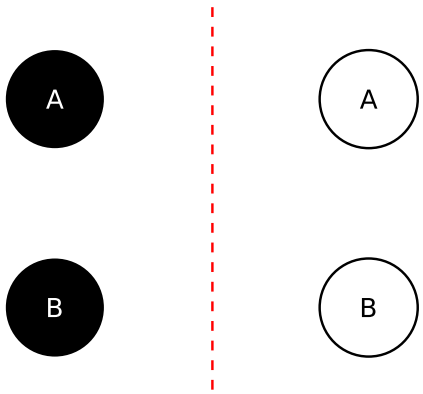


Figure 1.1: Objects grouped by color fill

1.1 there is an example illustrating the objects grouped by their color fill. It is important to notice that there are more possible groupings. For example, the objects can also be grouped by their corresponding letters. Which grouping is correct is highly dependent on the domain. The difficulty of verifying the

correctness of the result is one of the big challanges of clustering, and it is an important research topic on its own.

Formally, the task of clustering is stated as follows. Given a set of n data points $X = \{\vec{x}_1, \cdots, \vec{x}_n\}$, where each $\vec{x}_i$ is a d-dimensional vector, find the partitioning of data $R = \{P_1, \cdots, P_k\}$, consisting of k subsets of X. The points within each P_i should be as close as possible, and the points between P_i and P_j should be as far apart as possible.

There are several algorithms that can perform clustering [2]. Each algorithm has its own advantages and disadvantages, both in terms of produced results and computational resources required. It is highly dependent on the problem which algorithm to use. Connectivity and centroid models are a good example of commonly used clustering algorithms.

The connectivity models are based on a distance connectivity. An example of a connectivity model would be hierarchical clustering. The centroid models use the centroids in order to represent various clusters. An example of a centroid model would be the k-means clustering.

This thesis will focus only on the connectivity models, in particular, the agglomerative hierarchical clustering. This method is of interest because it is deterministic and it allows to observe the groupings at different levels of granuality.[1] Those are very important properties and are very useful in many areas of research. The biggest drawback of agglomerative hierarchical clustering is its high computational time complexity, which makes it inpractical [1] for large data sets. This is why it is important to explore various approximation methods that attempt to overcome the time complexity limitation, while preserving the good properties of the algorithm.

Chapter 2

Agglomerative hierarchical clustering

Agglomerative hierarchical clustering is a method of cluster analysis that aims to build a hierarchy of clusters. Instead of producing one clustering result, it gives several clustering partitions for different numbers of clusters. It is also flexible as it is possible to use various distance metrics within the algorithm [3], making it more suitable for a larger variety of problems.

This chapter presents agglomerative hierarchical clustering algorithm and some of its important properties and components such as distance metrics and linkage criteria. There is a dedicated section on its time and memory complexity, showing why the algorithm is not suitable for large data sets.

2.1 Algorithm

Given a data set $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$ of size n the algorithm produces an ordered sequence of cluster merges L, such that each element of L is a tuple consisting of clusters to be merged and the corresponding distance between the two clusters. The sequence L can be refered to as a linkage [3].

At the beginning of the algorithm every point is a cluster. Then, the pairwise distances between all clusters are computed. A pair of clusters with the smallest distance is merged into a single cluster. The process repeats until there is only one cluster left.

Algorithm 1 AHC(X)

$L \leftarrow [\,]$
$N \leftarrow |X|$
$S \leftarrow \{\text{index}(\vec{x})\}\ (\forall \vec{x} \in X)$ ▷ Get indices of all points
$D[a, b] \leftarrow d(\vec{x}_a, \vec{x}_b)\ (\forall a, b \in S)$ ▷ Compute all pairwise distances
for $i \leftarrow 0$ to $N - 2$ **do**
    $(a, b) \leftarrow \text{argmin}_{S \times S}(D)$ ▷ Find the closest points
    Append $(a, b, D[a, b])$ to L
    $S \leftarrow S \setminus \{a, b\}$
    $P_n \leftarrow P_a \cup P_b : n \notin S$ ▷ Merge the two clusters
    $D[n, c] \leftarrow l(P_n, P_c)\ (\forall c \in S)$ ▷ Update the distance matrix
    $S \leftarrow S \cup \{n\}$
    $R \leftarrow R \setminus \{P_a, P_b\}$
    $R \leftarrow R \cup \{P_n\}$
end for
return L

Algorithm 1 presents agglomerative hierarchical clustering in more detail. Different distance metrics and linkage criteria can be used within the algorithm. They are discussed in more detail in the following sections.

2.2 Distance metrics

Various metrics can be used to measure pairwise distances between the points. Each metric has its own advantages for a particular task and has the following properties :

1. $d(\vec{x}, \vec{y}) \geq 0$
2. $d(\vec{x}, \vec{y}) = d(y, x)$
3. $d(\vec{x}, \vec{y}) \leq d(\vec{x}, \vec{z}) + d(\vec{z}, \vec{y})$

This section presents some of the widely used distance metrics, and shows how each metric can be computed.

Euclidean distance
One of such metrics is the Euclidean distance. Given two points $\vec{x} = [x_1, \ldots, x_d]$ and $\vec{y} = [y_1, \ldots, y_d]$, the Euclidean distance is calculated as follows:

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}.$$

Euclidean squared distance
Euclidean squared distance is very similar to the Euclidean distance. The only difference is that it does not take the square root of the result. It can be calculated as follows:

$$d(\vec{x}, \vec{y}) = \sum_{i=1}^{d} (x_i - y_i)^2.$$

Manhattan distance
Another metric that can be used with the hierarchical clustering is called Manhattan distance and is calculated as follows:

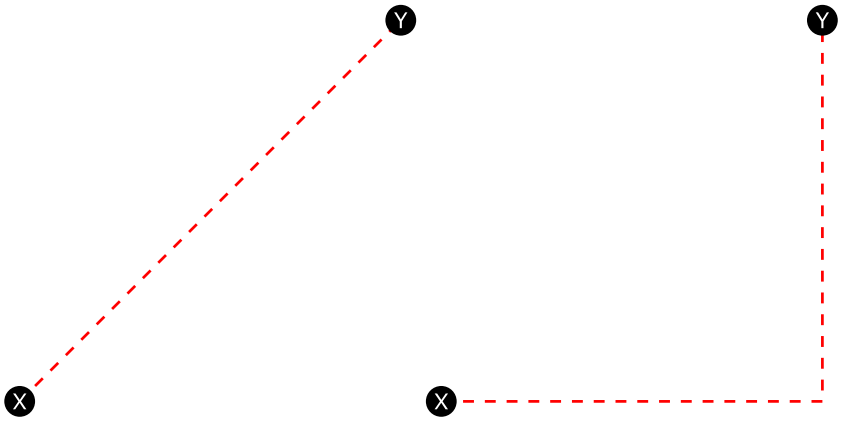
$$d(\vec{x}, \vec{y}) = \sum_{i=1}^{d} |x_i - y_i|.$$

Unlike the Euclidean distance, the Manhattan distance is measured at right angles along the axes. This difference is illustrated in Figure 2.1.

Minkowski distance
It is important to note that the Euclidean and Manhattan distances can be generalized as the p-norm which can be computed as follows:

$$\|\vec{x} - \vec{y}\|_p = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}.$$

For the Euclidean distance $p = 2$ and for the Manhattan distance $p = 1$.



(a) Euclidean distance (b) Manhattan distance

Figure 2.1: Visualization of Euclidean and Manhattan distances

Chebychev distance
The Chebychev distance [4] is defined as the maximum distance between the two points for any single dimension. It can be found as follows:

$$d(\vec{x}, \vec{y}) = \max_i |x_i - y_i|.$$

It is useful when the distances between the vectors are influenced more by some dimensions then the other.

Canberra distance
Canberra distance is very similar to Manhattan distance. The only difference is that there is a division by the sum of absolute values of two variables before the summation. Canberra distance can be calculated as follows:

$$d(\vec{x}, \vec{y}) = \sum_{i=1}^{d} \frac{|x_i - y_i|}{|x_i| + |y_i|}.$$

2.3 Linkage criteria

Linkage criteria have an important role in hierarchical clustering. They are used to determine how clusters will be merged during the algorithm. There are various linkage criteria that directly influence the outcome of clustering. Different linkage criteria use different strategies to determine the distances between two clusters P and P'.

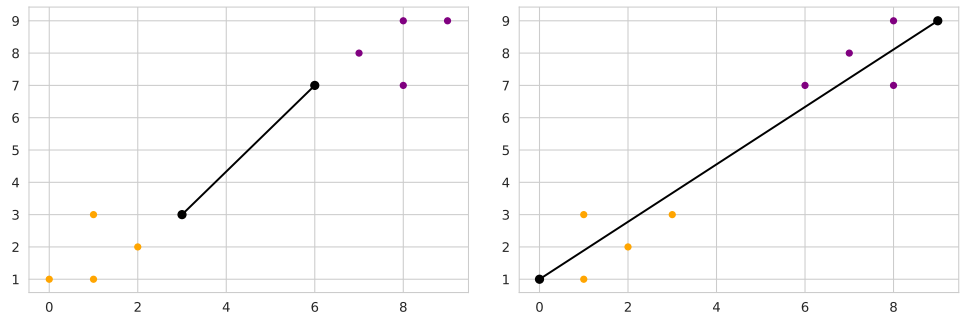
Single linkage
The single linkage [3] calculates the distance between two clusters as the distance between two closest points of those clusters. It can be calculated as follows:

$$l(P, P') = \min_{\vec{x} \in P, \vec{y} \in P'} d(\vec{x}, \vec{y}).$$

The distance between two points $d(\vec{x}, \vec{y})$ can be calculated using any distance metric presented in Section 2.2.

Complete linkage
The complete linkage [3] is similar to the single linkage. The only difference is that it uses points that are furthest from each other to determine the distance between two clusters. It uses the following distance:

$$l(P, P') = \max_{\vec{x} \in P, \vec{y} \in P'} d(\vec{x}, \vec{y}).$$



(a) Single linkage (b) Complete linkage

Figure 2.2: Single and complete linkage criteria

The difference between the two linkages can be seen on Figure 2.2.

Average linkage
Another popular approach is to use the average linkage [3], which uses the distance determined by calculating all pairwise point distances between two

clusters, and taking their average. The average linkage uses the following distance before each merge operation:

$$l(P, P') = \frac{1}{|P| \cdot |P'|} \sum_{\vec{x} \in P, \vec{y} \in P'} d(\vec{x}, \vec{y}).$$

Centroid linkage

The centroid linkage [3] uses the distance determined by the distance between the centers of each cluster. It can be found using the following formula:

$$l(P, P') = d\Big(\sum_{\vec{x} \in P} \frac{\vec{x}}{|P|}, \sum_{\vec{y} \in P'} \frac{\vec{y}}{|P'|} \Big).$$

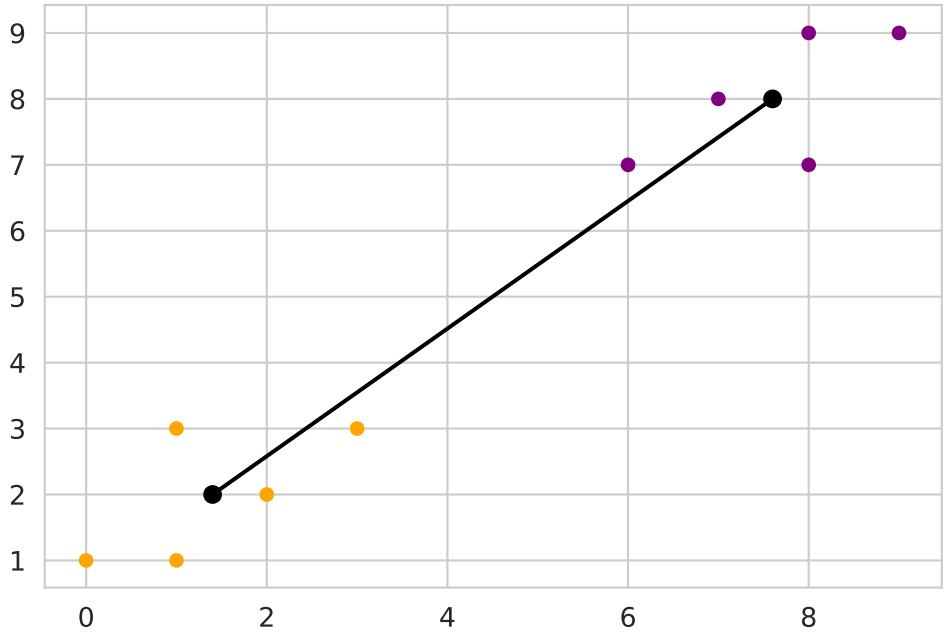


Figure 2.3: Centroid linkage

The centroid linkage can be seen in Figure 2.3.

2.4 Complexity

The agglomerative hierarchical clustering algorithm has many useful properties. It is deterministic and, it finds several clusters, at different levels of granuality, which are then presented in the form of a dendrogram. Unfortunately, the agglomerative hierarchical clustering algorithm is not scalable. It performs poorly on the large data sets. The biggest drawback of this algorithm is its time complexity.

In order to observe this limitation, it is important to analyze each step of the algorithm. The outermost loop of the algorithm performs n iterations of the merge operation, each having a constant time complexity. Before each merge operation, there is a need to recompute the distance matrix and find the smallest dissimilarities between any two clusters. This operation requires to compute all of the pairwise distances, and it has the time complexity of $\mathcal{O}(n^2)$. Pairwise distance computation is the most demanding part of the algorithm. It is easy to derive the overall time complexity of the algorithm. The outermost loop has the time complexity of $\mathcal{O}(n)$, and each iteration of the loop has the time complexity of $\mathcal{O}(n^2)$. Hence, the overall time complexity is $\mathcal{O}(n^3)$. Since there is a need to store the dissimilarity matrix, the memory complexity of the agglomerative hierarchical clustering algorithm is $\mathcal{O}(n^2)$. It is important to note that this time complexity is for the general form of the algorithm. There are methods that can reduce the time complexity for certain linkages, while still preserving the properties of the algorithm. For example, there is the MST-linkage algorithm [3], which is the algorithm for the single-linkage clustering, and it has the time complexity of $\mathcal{O}(n^2)$. The MST-linkage keeps the record of the smallest distances in an array, and does not require to recalculate the distance matrix at each iteration. Algorithm 2 shows the MST-linkage algorithm in more detail. It can be seen that there is an outer loop that performs $n-1$ iterations, and there are two inner loops. Each inner loop starts with n iterations, but this number decreases by one with each outer loop iteration of the algorithm. Therefore the time complexity of MST-linkage is $\mathcal{O}((n-1)n/2)$ which simplifies to $\mathcal{O}(n^2)$.

Algorithm 2 MST-linkage(X)

$L \leftarrow [\,]$
$N \leftarrow |X|$
$S \leftarrow \{\text{index}(\vec{x})\}\ (\forall \vec{x} \in X)$ ▷ Get indices of all points
$D[a, b] \leftarrow d(\vec{x}_a, \vec{x}_b)\ (\forall a, b \in S)$ ▷ Compute all pairwise distances
$c \leftarrow s\ (s \in S)$ ▷ Get any index from the set
$M[s] \leftarrow \infty (\forall s \in S \setminus \{c\})$ ▷ Keep record of minimum distances
for $i \leftarrow 1$ to $N - 1$ **do**
    $S \leftarrow S \setminus \{c\}$
    for each s in S **do**
        $M[s] \leftarrow \min(M[s], D[c, s])$ ▷ Update the minimum distances
    end for
    $n \leftarrow \text{argmin}_S(M)$ ▷ Get the closest point
    Append $(c, n, M[n])$ to L
    $c \leftarrow n$
end for
return L

2.5 Visualization

The results of hierarchical clustering can be visualized using a dendrogram. A dendrogram is a binary tree. Its leaves represent individual observations, and all other nodes represent the clusters, to which those observations belong to.

Such a dendrogram is shown in Figure 2.4. In this particular example, there are five observations A, B, C, D, and E. The vertical edges between the nodes show the distances between clusters. It can be seen that observations C and D are closer to each other than A and B. The dashed horizontal lines show some possible cuts of the dendrogram. The bottommost cut produces five clusters $\{A\}, \{B\}, \{C\}, \{D\}$, and $\{E\}$. The middle cut produces three clusters $\{A, B\}, \{C, D\}$, and $\{E\}$. Deciding where to cut the dendrogram, and what number of clusters to obtain is not an easy task. One of the ways it can be done, is by observing the dendrogram, and by finding such a level to cut, where the distances between the clusters, indicated by vertical lines, are relatively large.

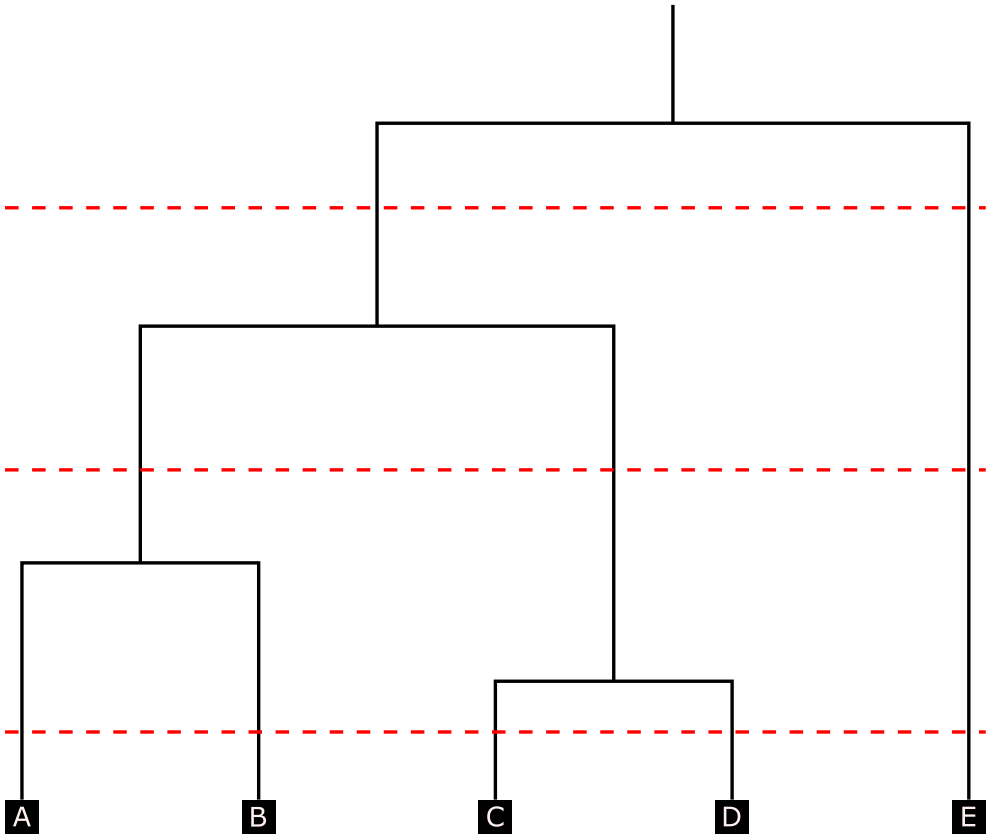


Figure 2.4: Dendrogram produced by agglomerative hierarchical clustering

2.6 Clustering evaluation

There are cases when it is necessary to measure how well a clustering algorithm partitioned the data. It is generally a hard problem since there are many ways in which the data can be partitioned. The correctness of a particular partition highly depends on the domain knowledge. There are several unsupervised performance metrics, that do not require the ground truth labels. Such metrics include the silhouette score [5] and Calinski-Harabaz index [6]. However, those metrics evaluate the performance purely in terms of general clustering definition. The score only depends on how close the points are within each cluster, and how far apart are individual clusters. There can be several partitionings that can give high unsupervised scores, but only a few would be useful for a particular domain.

Another way to evaluate the clustering results is to use the ground truth labels. The ground truth labels assign each element of the data set to one of the partitions. Given such labels, it is possible to evaluate how well the algorithm performed considering a specific domain. Three popular supervised

metrics are v-measure [7], adjusted rand index [8], and mutual-info score [8]. These metrics are functions that take two arguments, which are the true labels and the predicted labels. The output of each function is the score bounded by [-1.0, 1.0]. Scores that are close to 0.0 or negative indicate bad clustering, and scores that are close to 1.0 indicate good clustering. These metrics are symmetric, which means that if the argument positions are swapped, the score will still remain the same. The permutations are also taken into consideration, therefore it does not matter what label value is assigned to the points belonging to the same cluster. For example, the label assignment $[0, 0, 1, 1, 2]$ is equivalent to the label assignment $[2, 2, 0, 0, 1]$.

Silhouette score

The silhouette score directly evaluates the clustering results using the original data points. It does not require the true label assignment, hence it can be used for unsupervised clustering perfomance evaluation. It measures how close an object is to other objects in the same cluster, and how far it is from the other clusters. The silhouette score is bounded by $[-1, 1]$, where higher value indicates better clustering.

$$s(\vec{x}_i) = \frac{b(\vec{x}_i) - a(\vec{x}_i)}{\max(a(\vec{x}_i), b(\vec{x}_i))} \tag{2.1}$$

Equation 2.1 shows how to compute the silhouette score for a single point. The value of $a(\vec{x}_i)$ indicates how dissimilar a point is to its own cluster, therefore it is better if this value is small. The value of $b(\vec{x}_i)$ indicates how dissimilar a point is from the other clusters, hence it is better if this value is greater. The algorithm is as follows. For each point $\vec{x}_i$ in the data set X calculate the average distance to the points in the same cluster, and denote it as $a(\vec{x}_i)$, and then calculate average distances to other clusters, and denote the smallest distance as $b(\vec{x}_i)$. Now it is possible to calculate $s(\vec{x}_i)$ using the Equation 2.1. The final score is obtained by averaging all the values $s(\vec{x}_i)$ for all points.

Calinski-Harabaz index

Similarly to silhouette score, the Calinski-Harabaz score does not require the knowledge of true labels. It uses both the features and the clusters obtained to measure the perfomance. The greater the Calinski-Harabaz score is, the better the obtained clusters are. Unlike the silhouette score, it is not bounded. In order to obtain the score, it is first necessary to obtain between the group dispersion matrix B_k and within-cluster dispersion matrix W_k.

$$W_k = \sum_{q=1}^{k} \sum_{\vec{x} \in P_q} (\vec{x} - c_q)(\vec{x} - c_q)^T \tag{2.2}$$

$$B_k = \sum_{q} n_q (c_q - c)(c_q - c)^T \tag{2.3}$$

The matrices are obtained using the Equation 2.2 and Equation 2.3, where k is the number of clusters, $\vec{x} \in P_q$ is the point belonging to the cluster P_q, c_q is the center of the cluster P_q, n_q is the number of points in cluster P_q, and c is calculated as follows:

$$c = \sum_{q=1}^{k} \frac{n_q}{n} \cdot c_q.$$

Here, n is the total number of points in the data set.

$$s = \frac{\mathrm{Tr}(B_k)}{\mathrm{Tr}(W_k)} \times \frac{n-k}{k-1} \tag{2.4}$$

The final score is obtained using the Equation 2.4.

Adjusted Rand index

If the true labels are available, the adjusted Rand index can be used to calculate the similarity between two cluster assignments, such that the permutations are ignored. It takes two cluster assignments as an input and returns a score in the range $[-1, 1]$, where the higher the score, the more similar two cluster assignments are, which means better clustering if one of the cluster assignments is the true assignment. The adjusted Rand index is symmetric and it does not assume any clustering algorithm, hence it can be used with various clustering algorithms. It is suitable for small sample sizes or large numbers of clusters. In order to obtain the adjusted Rand index it is necessary to construct the contingency table. In Table 2.1 $\{P_1, \cdots, P_r\}$ and $\{P'_1, \cdots, P'_s\}$

	P'_1	P'_2	$\cdots$	P'_s	Sum
P_1	n_{11}	n_{12}	$\cdots$	n_{1s}	a_1
P_2	n_{21}	n_{22}	$\cdots$	n_{2s}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
P_r	n_{r1}	n_{r2}	$\cdots$	n_{rs}	a_r
Sum	b_1	b_2	$\cdots$	b_s	n

Table 2.1: Contingency table

represent true and predicted cluster assignments. For example, P_1 represents the set of elements belonging to the true cluster 1, and each $n_{ij} = |P_i \cap P'_j|$

represents the number of elements contained in both P_i and P'_j.

$$s = \frac{\sum_{ij} \binom{n_{ij}}{2} - \frac{\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}}{\binom{n}{2}}}{\frac{\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}}{2} - \frac{\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}}{\binom{n}{2}}} \tag{2.5}$$

Once the contingency table is constructed, the adjusted Rand index can be obtained by using the Equation 2.5.

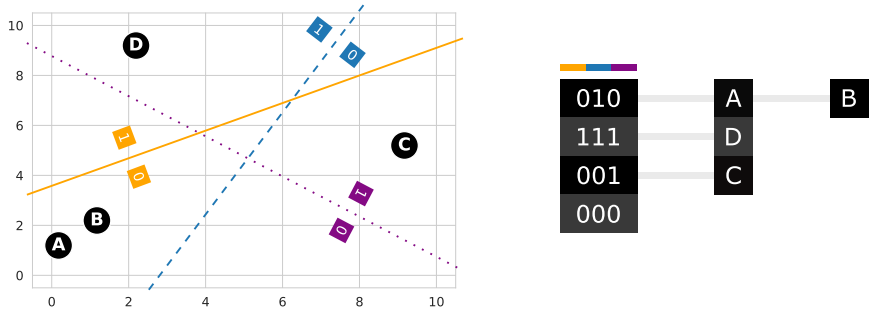
Chapter 3

Approximate methods

The biggest drawback of hierarchical clustering is its time complexity [1]. Since the algorithm has to evaluate all pairwise distances between the points, it is not suitable for large or even medium data sets. On the other hand it has some good properties when compared to other algorithms such as k-means, and it is used for many practical applications. Approximate hierarchical clustering is one of the ways to address the problem of complexity. There exist several approximate methods, and each has its own advantages and disadvantages. This chapter will present and compare some of those methods.

3.1 LSH-link

The LSH-link [1] is an approximate algorithm for the agglomerative hierarchical clustering with the single-linkage and it has the time complexity of $\mathcal{O}(nB)$. It uses the idea of locality-sensitive hashing [9] to achieve better performance, while producing similar results to the standard algorithm. Locality-sensitive hashing is an approximate nearest-neighbor search method that attempts to reduce the curse of dimensionality. It uses hash functions that map points in multidimensional space into a single scalar value, and try to store similar points into a single hash entry. For that purpose, locality-sensitive hashing maximizes the collisions for the objects that are close to each other. As a result the relative distances between the original points are preserved after the hash operation. This property is very useful in the context of hierarchical clustering. The standard hierarchical clustering algorithm requires to compute all of the pairwise distances between the points. It equally considers the points that are actually close to each other, and the points that are far apart. Locality-sensitive hashing speeds up the pairwise computation by considereng only the points that are more likely to be similar, and this is the key idea behind the LSH-link algorithm. There can be different hash functions that can project the points from higher dimensional spaces to lower dimensional spaces and still preserve the relative distances. One of such hash functions utilizes



(a) Points in a two dimensional space (b) Hash table created from the points

Figure 3.1: Points that are close to each other are in the same bucket of the hash table

the random projection [9]. Let $\vec{x}$ be a point in a multi-dimensional space. In order to obtain a hash value it is necessary to perform dot products between that point and a sequence of vectors $V = [\vec{v}_1, \ldots, \vec{v}_k]$. The components of each vector $\vec{v}_i$ are drawn from the normal distribution. A sequence of ones and zeros, which is the final hash value, is produced as follows:

$$h^k(\vec{x}) = [\vec{x} \cdot \vec{v}_1 > 0, \ldots, \vec{x} \cdot \vec{v}_k > 0].$$

For example, in the Figure 3.1 there are 4 points. The solid orange line represents the hyperplane given by the equation $\vec{x}_i \cdot \vec{v}_1 > 0$, the dashed blue line represents the hyperplane given by the equation $\vec{x}_i \cdot \vec{v}_2 > 0$, and the dotted purple line represents the hyperplane given by the equation $\vec{x}_i \cdot \vec{v}_3 > 0$. The hyperplanes are used to partition the space, which makes it possible to obtain a hash value for each point. The hash value for the point $\vec{A}$ is obtained by performing the dot product with each of the vectors $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$ which results in 010. In general, for k hyperplanes, there will be 2^k regions. It can be observed that k is the length of the hash value. This hash function is different from the one presented in the original LSH-link paper [1], which was used for the demonstration purposes and worked only with integer data sets. The hash function specified above works with real valued variables. The fact that vectors of V are randomly sampled, means that there is always a chance that similar points might not actually end up in the same bucket of a hash table. In order to overcome this problem it is necessary to create several hash functions, each having its own V. When the hash tables are created using the corresponding hash functions, the unions between the point related buckets are performed. The idea behind multiple hash tables is that it is not very likely that similar points will end up in different buckets for all of the hash tables. However, if the number of hash tables is too big then there is a higher chance that points which are far appart can also end up in the same bucket.

Therefore, the number of hash tables is also a parameter that needs to be optimized depending on a problem.

Given the knowledge of the locality-sensitive hashing it is now possible to describe the LSH-link algorithm. At each iteration of the algorithm l hash functions are generated. Each point is then inserted into l hash tables unless there is already a point from the same cluster in the same bucket. In order to get all of the neighboring points to a point of interest, the union of the corresponding buckets is performed. The actual distances between the point and its neighbors are then computed. If the distance between the two points is below the specified distance threshold r then the clusters corresponding to these points are merged. The algorithm terminates when there is only one cluster left. Initially, the distance r should be selected to be relatively small, so that close points are merged first. At each iteration r is increased by a factor of A in order to ensure that distant points are eventually merged into one cluster. The value of A is also a parameter of the algorithm. Smaller values of A will produce closer approximations at the cost of higher computational cost. The value k is decreased in each iteration in order to increase the collision between more distant points. Algorithm 3 presents the LSH-link in more details.

Algorithm 3 LSH-link(X, A, r)

$L \leftarrow [\,]$
$d \leftarrow \dim(X)$
$C \leftarrow \max(X)$
$S \leftarrow \{\text{index}(\vec{x})\}\ (\forall \vec{x} \in X)$ ▷ Get indices of all points
while $|R| \neq 1$ **do**
    $k \leftarrow dC\sqrt{d}/2r$ ▷ Update the value k
    $T \leftarrow \{t_1, \ldots, t_l\}$ ▷ Generate l hash tables
    Insert $\vec{x}$ into t $(\forall \vec{x} \in X, \forall t \in T)$
    for each a in S **do**
        $M \leftarrow \bigcup_{t \in T} t[\vec{x}_a]$ ▷ Get all potential neighbors
        for each b in M **do**
            if $d(\vec{x}_a, \vec{x}_b) \leq r$ and $P_a \neq P_b$ **then**
                $P_a \leftarrow P_a \cup P_b$ ▷ Merge the two clusters
                $R \leftarrow R \setminus \{P_b\}$
                Append $(a, b, d(\vec{x}_a, \vec{x}_b))$ to L
            end if
        end for
    end for
    $r \leftarrow r * A$
end while
return L

3.1.1 Complexity

It is important to show that the time complexity of the LSH-link algorithm is $\mathcal{O}(nB)$. Here, the value B is the maximum number of points that can enter a single bucket of a hash table [1]. There is a dependence of B on n, since n points are distributed in the hash tables. In order to understand the overall time complexity it is important to analyze each part of the algorithm individually. The algorithm consits of three major parts: generation of hash tables, finding the clusters to be merged, and updating the clusters. In the hash table generation phase it is necessary to create l hash tables. Each point is hashed through the l hash tables. In total there are n points. The application of the hash function has a constant time complexity. Hence, the time complexity of hash table generation is $\mathcal{O}(nl)$ which simplifies to $\mathcal{O}(n)$ since l is a constant independent of n. The next phase of the algorithm is to find the clusters to be merged. That operation requires to compute the distance between each point and its corresponding l buckets that can contain at most B elements. Hence the time complexity of such operation is $\mathcal{O}(nlB)$. As in the previous step, l is a constant independent of n, therefore, the time complexity of finding the clusters to be merged is $\mathcal{O}(nB)$. The last part of the algorithm requires to update the clusters, that is to merge clusters together. Each cluster can be represented as a vertex in a graph. The two clusters which are to be merged will have an edge between them. The problem of merging the clusters can be seen as the graph decomposition into connected components, which is achieved in $\mathcal{O}(N + M)$, where N is the number of vertices and M is the number of edges. In this case N is bounded by n, and M is bounded by nlB. Therefore the time complexity of finding the connected components is $\mathcal{O}(n+nlB)$, which simplifies to $\mathcal{O}(nB)$. Therefore the overall time complexity of the LSH-link algorithm is $\mathcal{O}(n + nB + nB)$, which simplifies to $\mathcal{O}(nB)$.

3.2 Hybrid hierarchical clustering

The hybrid hierarchical clustering algorithm [10] attempts to overcome the time complexity overhead by using the combination of the partitional k-means algorithm [11] and the standard agglomerative hierarchical clustering algorithm. It is common to perform clustering using the k-means algorithm due to its linear time complexity and simplicity. Given the data set X of n observations, where each observation is a d-dimensional vector, the k-means algorithm starts with the random initialization of k centroids, which are essentially additional d-dimensional vectors. Each centroid corresponds to one of the k clusters. At the second step the Euclidean distances between each point and the k centroids are computed. Each point is then assigned to its closest centroid. At the next step the positions of the centroids are updated. The new position of each centroid is obtained by computing the mean of its corresponding points. If there was a change in the position of the centroids and it is not the last iteration, as specified by the user, the algorithm continues from the second step. Otherwise, the algorithm terminates. The hybrid hierarchical clustering uses the k-means algorithm in order to create k partitions $P_1, \ldots, P_k$ of the original data set. The greater the value k the closer the hybrid algorithm will approximate the standard agglomerative hierarchical clustering algorithm. In order to achieve the running time of $\mathcal{O}(n\sqrt{n})$, k should be picked close to $\sqrt{n}$ [10]. The exact hierarchical clustering is then performed on each partition P_i in order to create the trees $T_1, \ldots, T_k$. This step can be referred to as the first level hierarchical clustering. The obtained trees are then treated as nodes during the second level hierarchical clustering. This means that all of the trees are clustered into a single tree T, which is the final result of the algorithm.

Figure 3.2 shows how points are initially partitioned using the k-means algorithm, and then each partition is clustered using the first level hierarchical clustering to create trees. Figure 3.3 shows how the obtained trees are clustered into a single tree using the second level hierarchical clustering. The difference between the first and second levels of clustering is that the first level is performed on the points and the second level is performed on the partitions. The Algorithm 4 shows the hybrid hierarchical clustering algorithm in more detail. It can be observed that there is recursion involved in one of the steps. It is required for the case when the number of points in a partition is higher then the specified threshold t, which can slow down the exection of the agglomerative hierarchical clustering.

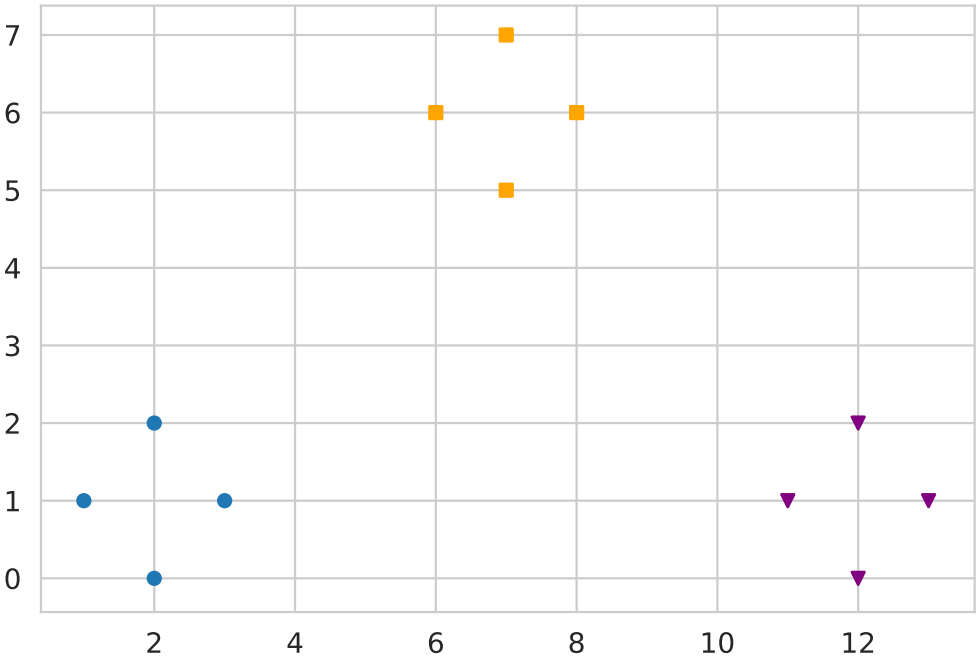
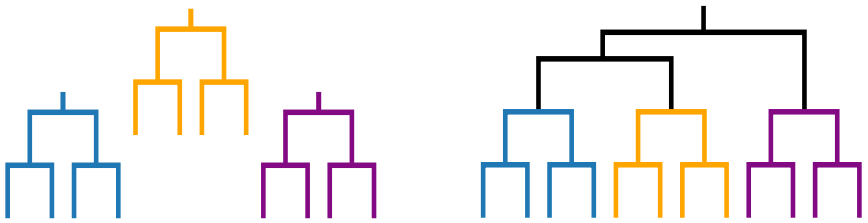


Figure 3.2: Objects are first clustered using the k-means algorithm resulting in three partitions



(a) First level hierarchical clustering (b) Second level hierarchical clustering

Figure 3.3: Trees obtained by the first level hierarchical clustering are merged into a single tree using the second level hierarchical clustering

3.2.1 Complexity

The time complexity of the hybrid hierarchical clustering can be $\mathcal{O}(n\sqrt{n})$. It highly depends on the parameter k. For larger values of k the algorithm will perform slower, but the obtained results will closer resemble the ones produced by the exact hierarchical clustering algorithm. The overall time complexity is affected by two parts of the algorithm. The first part is the partitioning of data using the k-means algorithm, which has the time complexity of $\mathcal{O}(knI)$,

Algorithm 4 HHC(X)

```
1: R ← k-means(X)
2: for each P_i in R do
3:     r_i ← argmin_j l(P_i, P_j)
4:     for each p in P_i do
5:         if d(p, C_i) > r_i then
6:             P_i ← P_i \ p
7:             R ← R ∪ {p}
8:         end if
9:     end for
10: end for
11: for each P_i in R do
12:     if |P_i| > t then
13:         T_i ← HHC(P_i)
14:     else
15:         T_i ← AHC(P_i)
16:     end if
17: end for
18: T ← AHC({T_1, ..., T_m})
19: return T
```

where k is the number of centroids, n is the number of observations and I is the number of iterations. The second part is the agglomerative hierarchical clustering of the obtained partitions. There are k partitions, each of size approximately n/k. Assuming the quadratic time complexity of agglomerative hierarchical clustering, the time complexity of clustering each partition is $\mathcal{O}(n^2/k)$. The time complexity of clustering k trees into a single tree is $\mathcal{O}(k^2)$. Therefore the overall time complexity is $\mathcal{O}(knI + n^2/k + k^2)$. If k is set to approximately $\sqrt{n}$, then the time complexity is $\mathcal{O}(n\sqrt{n}I + n\sqrt{n} + n)$ which simplifies to $\mathcal{O}(n\sqrt{n})$ since I is a small constant value.

Chapter 4

Experiment

This chapter presents the experimental part of the thesis. The previous chapter showed the theoretical time complexity improvements of the approximation methods. It is important to show that those improvements also hold in practice. In order to do this, it is necessary to test one of proposed methods on the real data sets, and compare it with the exact agglomerative hierarchical clustering algorithm. The experiment setup will be described first. It will present what quantities will be measured and how they will be measured and compared. Then, there will be a description of each data set used, and the motivation behind choosing each data set. The implementation details of the used algorithms will be described as well as the motivation behind implementing each algorithm. The algorithms will be run and compared on the real data sets, and the results will be presented.

4.1 Experiment design

In this experiment two algorithms will be compared. The first algorithm is the agglomerative hierarchical clustering using the single linkage called MST-linkage. The second algorithm is the approximate method called the LSH-link. It is important to note that LSH-link is an approximation method for the single linkage. The MST-linkage algorithm will provide the reference results that will help evaluate the results obtained by the LSH-link algorithm. The goal of the experiment is to show that the approximation method indeed performs faster than the standard algorithm, while still producing similar results.

In order to quantatively compare the approximation method to the exact agglomerative hierarchical clustering it is important to specify how the clustering quality and the running time will be measured. To achieve a fair comparison, all of the algorithms will be run on the same machine, and the same data sets will be used throughout the experiment. There are several methods to verify the validity of the clustering results.

The first way is to use supervised performance metrics for clustering. Such metrics require the ground truth labels. There are several metrics that can be used, and their selection is dependent on the problem and its underlying domain. Such metrics were presented in Section 2.6. Both algorithms need to run on the same data set and produce the two linkages. If a data set has n points, then the linkage will have n levels. Both linkages are then iterated over each level simultaneously. At each level, both linkages are cut in order to produce two flat cluster assignments. Each flat cluster assignment produced by the MST-linkage will serve as the true label assignment, and each flat cluster assignment produced by the LSH-link will serve as the predicted label assignment. The supervised metrics are then used to obtain the scores between each pair of such assignments. As a result, there will be n scores, each corresponding to the specific cut of the linkage. The final score is the median of the n scores. The intuition behind this approach is that if the produced linkages are similar, then the flat clusters at each level should also be similar, and this similarity can be measured by using the supervised metrics.

The second way is to produce a dendrogram using the exact agglomerative hierarchical clustering algorithm and then visually compare it to a dendrogram produced by the approximation methods. Visual similarity is not a reliable way to evaluate the performance, however it can give an insight on the overall structure of the produced results.

After the clustering performance is evaluated, both algorithms will ran on large data sets to show the difference in running time performance. One of the ways to achieve this is to split a large data set into a sequence of data sets, where each consequent data set has a greater number of data points. Then, for each such data set in the sequence, both algorithms will run 10 times, and the average execution times will be recorded. The obtained execution times can then be plotted, which should show the difference between the two algorithms in terms of running time with respect to data size.

Another important factor influencing the experiment is the selection of the data sets. Several real world data sets were chosen to be used with the clustering. The first data set is called Iris. It is useful for the purpose of dendrogram visualization, since it has few points. Another data set is called Sonar. It is of intereset because of its higher number of features, which can influence the running time of the algorithm. The third data set is called Glass. The Digits and Spam data sets were used as well, mainly because they both have large amount of samples and features. In order to measure the execution time of each algorithm, it is necessary to have a large data set. Such data set was generated from the multivariate normal distribution with the variance 1, but different means to represent different clusters. The data sets are summarized in Table 4.1.

Dataset	Number of instances	Number of features
Iris	150	4
Sonar	208	60
Glass	214	9
Spam	4601	57
Digits	1593	256
Artificial	10000	2

Table 4.1: Real world data sets used in the experiment

4.2 Implementation

Both algorithms used in this experiment were implemented in the C++ programming language. At the time of writing this thesis, there was no publicly available implementation of the LSH-link algorithm, therefore it was necessary to implement it in order to run the experiments. On the other hand, there are some implementations of the MST-linkage algorithm, however they have different interfaces. Both algorithms were implemented to share a common interface in terms of inputs and outputs. They both require a data set as an input and they both produce a linkage matrix that can be used to visualize a dendrogram, or to get the cluster assignments at any level of that dendrogram. For that reason, both algorithms confirm to the linkage interface specified by the SciPy scientific and technical computing library.

4.3 Results

This section presents the results of the experiments. The details of the experiment design were presented in Section 4.1.

4.3.1 Supervised scores

In order to evaluate the clustering performance of the algorithm it is necessary to use quantative measures such as the supervised clustering performance metrics. Table 4.2 shows the median performance scores obtained for each data

	V-measure	Adjusted Rand index	Mutual information
Iris	0.90	0.57	0.61
Sonar	0.85	0.58	0.48
Glass	0.91	0.58	0.57
Spam	0.79	0.44	0.44
Digits	0.83	0.22	0.36

Table 4.2: Supervised clustering performance scores for each data set

set. It can be seen that the clustering assignments agree the most according to the V-measure score. The V-measure score is the harmonic mean between homogeneity and completeness. Homogeneity means that each cluster contains only members of a single class. Completeness means that all members of a given class are assigned to the same cluster. However, the V-measure will not yield zero scores for random labelings if the number of clusters is above 10, and the sample sizes is below 1000 [7]. Therefore, the adjusted Rand index and the adjusted mutual information scores should also be taken into consideration. For these data sets the adjusted Rand scores and the adjusted mutual information scores are lower then the V-measure scores. This might be due to the LSH-link hyperparameters not being optimized for these particular data sets.

4.3.2 Visual comparison

In order to show the similarity between the produced dendrograms, both algorithms were run on the Iris data set. Even though this data set is very small, it serves well for the purpose of dendrogram visualization. In Figures

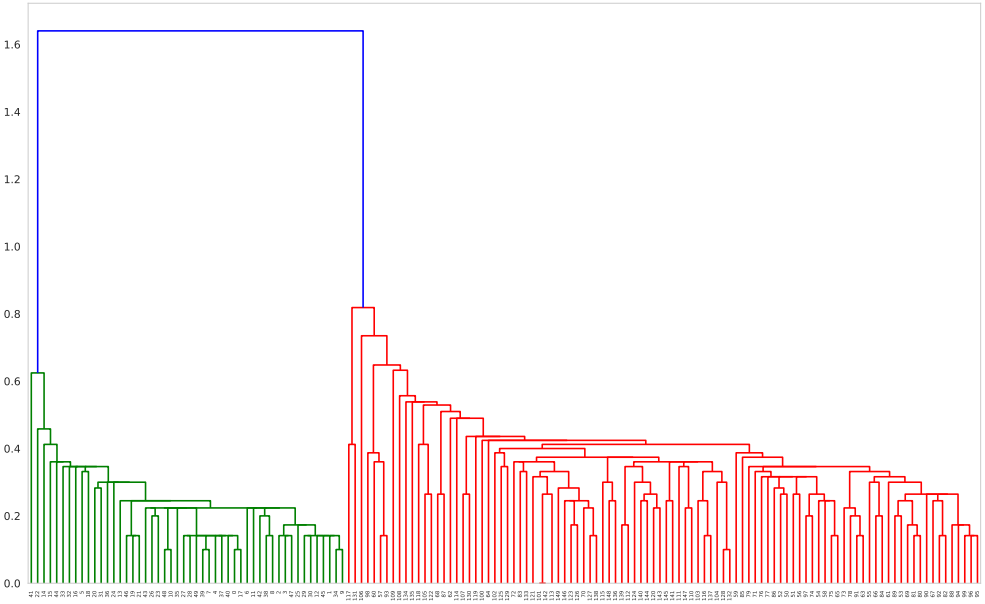


Figure 4.1: MST-linkage dendrogram

4.1 and 4.2 are the dendrograms produced by MST-linkage and LSH-link. It can be observed that the overall shapes of the dendrograms are similar. Visualization of dendrograms is crucial for many practical problems, therefore it is important to show that the dendrograms produced by LSH-link are of a similar structure to those produced by the single-linkage algorithm. It is

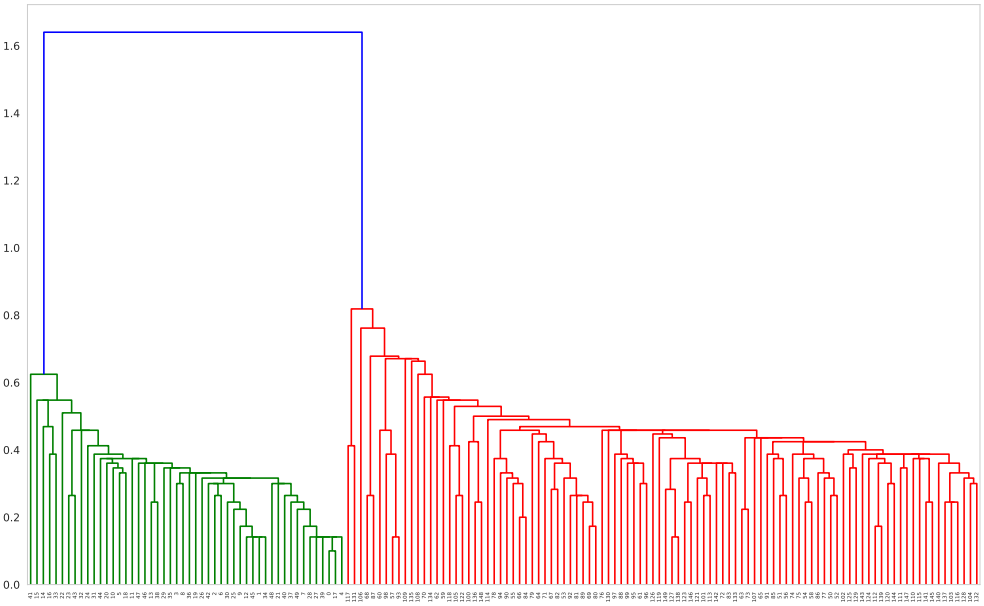


Figure 4.2: LSH-linkage dendrogram

important to note that visual comparison is not a reliable way to measure the clustering performance of the algorithm.

4.3.3 Execution time

One of the main goals of this thesis is to show that the approximation method performs faster then the the standard method. For each data set, the algorithms were ran several times, and the average execution times in milliseconds were recorded. The number of hash tables l was set to 1. The plots in Figures

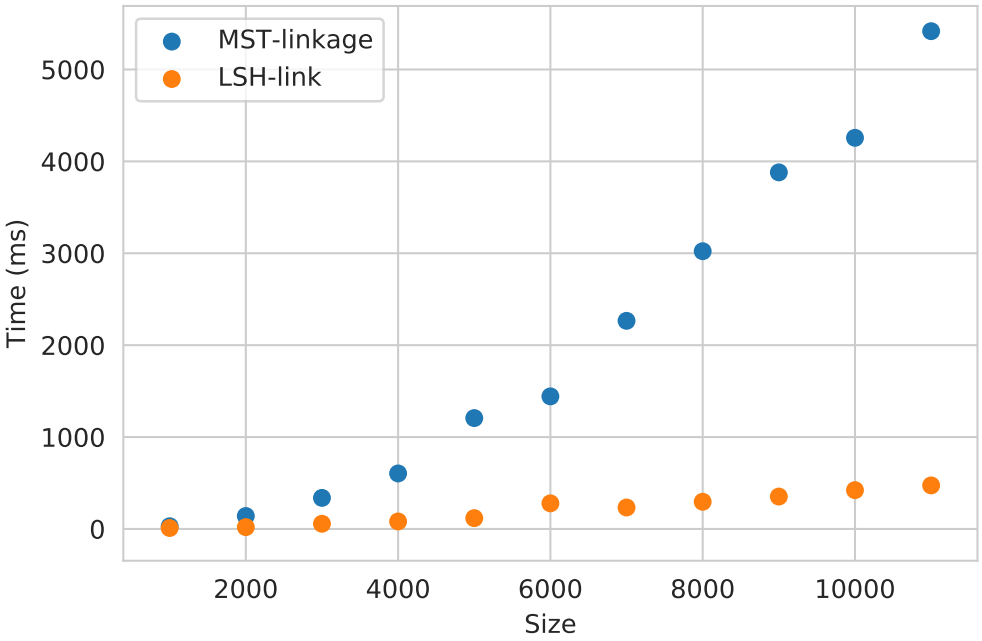


Figure 4.3: Execution time for artificial data set with respect to the number of data points

4.3, 4.4, and 4.5 show that as the number of points increases, the approximation algorithm performs significantly faster then the exact algorithm.

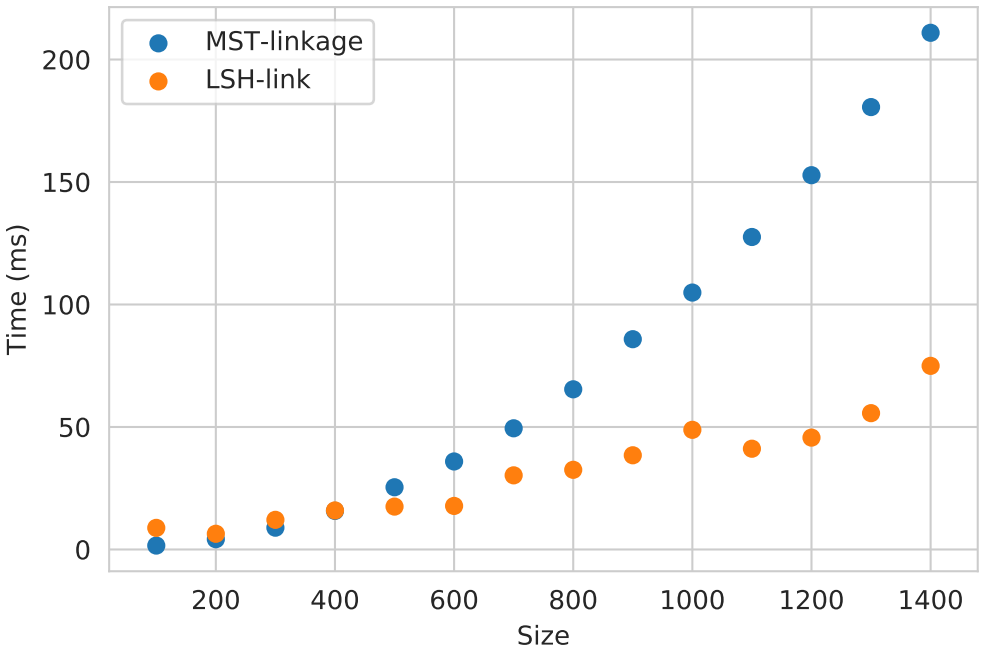


Figure 4.4: Execution time for Spam data set with respect to the number of data points

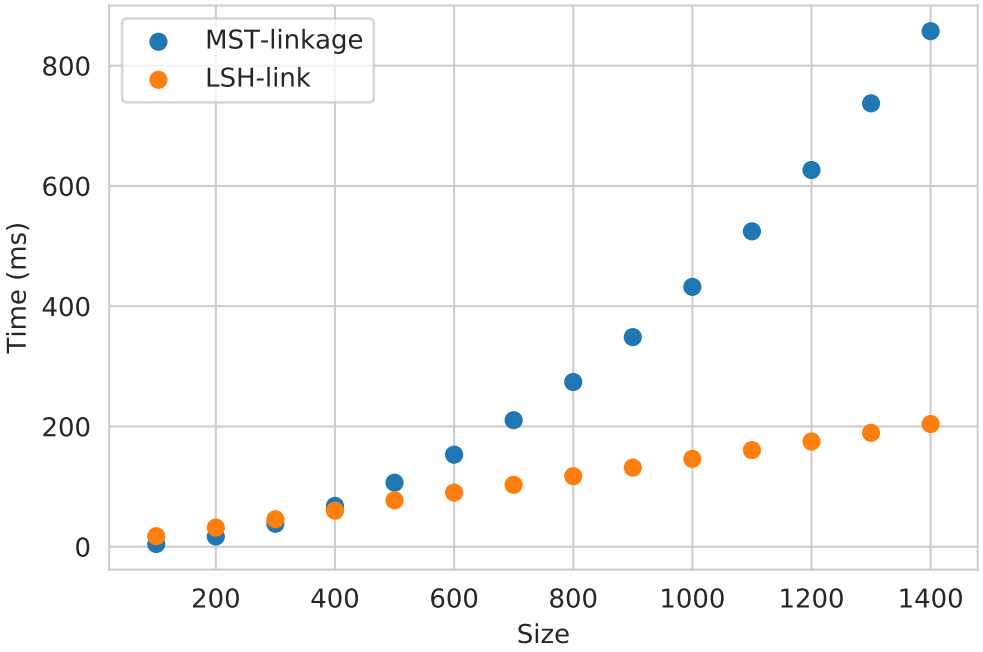


Figure 4.5: Execution time for Digits data set with respect to the number of data points

Conclusion

The general clustering problem, as well as the agglomerative hierarchical clustering algorithm, which is a determininstic algorithm that obtains a hierarchy of clusters, were introduced. It was shown that the agglomerative hierarchical clustering algorithm suffers from high time complexity of $\mathcal{O}(n^3)$, and is not suitable for large data sets. The algorithm components such as distance metrics, and the linkage criteria were presented and explained in detail. It was also shown how it is possible to evaluate the clustring results using the supervised metrics, and how to interpret the visualization using the dendrograms. Agglomerative hierarchical clustering approximation methods, that have lower time complexity then the exact algorithm, were explained, and the corresponding algorithms were analyzed in terms of their time complexities.

Two of the algorithms were implemented. The first implemented algorithm is the exact hierarchical clustering algorithm for the single linkage called MST-linkage with the time complexity of $\mathcal{O}(n^2)$. The second implemented algorithm, is an approximation algorithm LSH-link, that uses the nearest neighbor search method called locality sensitive hashing. It was shown that the time complexity of LSH-link is $\mathcal{O}(nB)$.

The implemented algorithms were ran on both real world and artificially generated data sets. The linkages produced by the two algorithms were compared using the supervised metrics. In terms of running time, it was shown that the LSH-link algorithm performed significantly faster then the MST-linkage algorithm, which supports the statement that it has lower time complexity.

Bibliography

[1] Koga, H.; Ishibashi, T.; Watanabe, T. Fast Hierarchical Clustering Algorithm Using Locality-Sensitive Hashing. 10 2004, pp. 114–128, doi: 10.1007/978-3-540-30214-8_9.

[2] Estivill-Castro, V. Why So Many Clustering Algorithms: A Position Paper. *SIGKDD Explor. Newsl.*, volume 4, no. 1, June 2002: pp. 65–75, ISSN 1931-0145, doi:10.1145/568574.568575. Available from: `http://doi.acm.org/10.1145/568574.568575`

[3] Müllner, D. Modern hierarchical, agglomerative clustering algorithms. 09 2011.

[4] Kirdat, T.; Patil, V. V. INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN TECHNOLOGY 28 Application of Chebyshev Distance and Minkowski Distance to CBIR Using Color Histogram. Technical report, 2016.

[5] Rousseeuw, P. Rousseeuw, P.J.: Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. Comput. Appl. Math. 20, 53-65. *Journal of Computational and Applied Mathematics*, volume 20, 11 1987: pp. 53–65, doi:10.1016/0377-0427(87)90125-7.

[6] Caliński, T.; JA, H. A Dendrite Method for Cluster Analysis. *Communications in Statistics - Theory and Methods*, volume 3, 01 1974: pp. 1–27, doi:10.1080/03610927408827101.

[7] Rosenberg, A.; Hirschberg, J. V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure. *Proceedings of the 2007 Joint Conference on Emprical Methods in Natural Language Processing and Computational Natural Language Learning*, , no. June, 2007: pp. 410–420, ISSN 1524-4040, doi:10.7916/D80V8N84.

[8] Vinh, N. X.; Epps, J.; Bailey, J. Information theoretic measures for clusterings comparison. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, volume 11, 2009: pp. 1–8, ISSN 15280020, doi:10.1145/1553374.1553511.

[9] Slaney, M.; Casey, M. Locality-Sensitive Hashing for Finding Nearest Neighbors [lecture NOTES] IEEE SIGNAL PROCESSING MAGAZINE [128]. 2008, doi:10.1109/MSP.2007.914237.

[10] Tanaseichuk, O.; Khodabakshi, A.; Petrov, D.; et al. An Efficient Hierarchical Clustering Algorithm for Large Datasets. *Austin J Proteomics Bioinform & Genomics. Austin J Proteomics Bioinform & Genomics*, volume 2, no. 2, 2015: pp. 1008–1, ISSN 2471-0423. Available from: `http://austinpublishinggroup.com/proteomics-bioinformatics-genomics/fulltext/ajpbg-v2-id1008.pdf`

[11] MacKay, D. J. C. An Example Inference Task: Clustering. *Information Theory, Inference and Learning Algorithms*, 2003: pp. 284–292, ISSN 0028-0836, doi:10.1038/nature10736, `NIHMS150003`.

Appendix A

Acronyms

LSH Locality-sensitive hashing

MST Minimum spanning tree

AHC Agglomerative hierarchical clustering

HHC Hybrid hierarchical clustering

Appendix B

Contents of enclosed USB flash drive

- `readme.txt` ..... the file with USB flash drive contents description
- `src` ..... the directory of source codes
  - `implementation` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format