Discretize and Conquer: Scalable Agglomerative Clustering in Hamming Space

by

Soheil Soltani

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Masters of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2018

Abstract

Clustering is one of the most fundamental tasks in many machine learning and information retrieval applications. Roughly speaking, the goal is to partition data instances such that similar instances end up in the same group while dissimilar instances lie in different groups. Quite surprisingly though, the formal and rigorous definition of clustering is not at all clear mainly because there is no consensus about what constitutes a cluster. That said, across all disciplines, from mathematics and statistics to genetics, people frequently try to get a first intuition about the data through identifying meaningful groups. Finding similar instances and grouping them are two main steps in clustering, and not surprisingly, both have been the subject of extensive study over recent decades.

It has been shown that using large datasets is the key to achieving acceptable levels of performance in data-driven applications. Today, the Internet is a vast resource for such datasets, each of which contains millions and billions of high-dimensional items such as images and text documents. However, for such large-scale datasets, the performance of the employed machine-learning algorithm quickly becomes the main bottleneck. Conventional clustering algorithms are no exception, and a great deal of effort has been devoted to developing scalable clustering algorithms. Clustering tasks can vary both in terms of the input they have and the output that they are expected to generate. For instance, the input of a clustering algorithm can hold various types of data such as continuous numerical, and categorical types. This thesis on a particular setting; in it, the input instances are represented with binary strings. Binary representation has several advantages such as storage efficiency, simplicity, lack of a numerical-data-like concept of noise, and being naturally normalized.

The literature abounds with applications of clustering binary data, such as in marketing, document clustering, and image clustering. As a more-concrete example, in marketing for an online store, each customer's basket is a binary representation of items. By clustering customers, the store can recommend items to customers with the same interests. In document clustering, documents can be represented as binary codes in which each element indicates whether a word exists in the document or not. Another notable application of binary codes is in *binary hashing*, which has been the topic of significant research in the last decade. The goal of binary hashing is to encode high-dimensional items, such as images, with *compact* binary strings so as to preserve a given notion of similarity. Such codes enable extremely fast *nearest neighbour searches*, as the distance between two codes (often the *Hamming distance*) can be computed quickly using bit-wise operations implemented at the hardware level.

Similar to other types of data, the clustering of binary datasets has witnessed considerable research recently. Unfortunately, most of the existing approaches are only concerned with devising density and centroid-based clustering algorithms, even though many other types of clustering techniques can be applied to binary data. One of the most popular and intuitive algorithms in *connectivity-based* clustering is the *Hierarchical Agglomerative Clustering* (HAC) algorithm, which is based on the core idea of objects being more related to nearby objects than to objects farther away. As the name suggests, HAC is a family of clustering methods that return a *dendrogram* as their output: that is, a hierarchical tree of domain subsets, having a singleton instance in their leaves and the whole data instances in their root. Such algorithms need no prior knowledge about the number of clusters. Most of them are deterministic and applicable to different cluster shapes, but these advantages come at the price of high computational and storage costs in comparison with other popular clustering algorithms such as k-means.

In this thesis, a family of HAC algorithms is proposed, called *Discretized Agglomerative Clustering* (DAC), that is designed to work with binary data. By leveraging the discretized and bounded nature of binary representation, the proposed algorithms can achieve significant speedup factors both in theory and practice, in comparison to the existing solutions. From the theoretical perspective, DAC algorithms can reduce the computational cost of hierarchical clustering from cubic to quadratic, matching the known lower bounds for HAC. The proposed approach is also be empirically compared with other well-known clustering algorithms such as k-means, DBSCAN, average, and complete-linkage HAC, on well-known datasets such as TEXMEX, CIFAR-10 and MNIST, which are among the standard benchmarks for large-scale algorithms. Results indicate that by mapping real points to binary vectors using existing binary hashing algorithms and clustering them with DAC, one can achieve several orders of magnitude speed without losing much clustering quality, and in some cases, achieving even more.

Acknowledgements

I would like to thank my supervisor, Dr. Ladan Tahvildari, for her constant support and feedback throughout my research. I am pleased to have joined a thesis-based Masters under her supervision; it has indeed changed my perspective on academia. It has been a pleasure working with Ladan, who was always there to help, not only in research-related matters but also as a friend.

Thanks are also owed to the members of my dissertation committee for taking time out of their busy schedules to read my thesis and provide me with their insightful suggestions: Dr. Otman Basir for his encouraging comments and indispensable support, and Dr. Mark Crowley for his inspiring remarks that broadened my perspectives on research.

I would also like to especially thank Sepehr Eghbali for spending his valuable time to read my thesis, provide feedback, and make helpful comments. I would thank my colleagues in the Electrical and Computer Engineering, and the Computer Science departments of the University of Waterloo: Dr. Mahsa Emami Taba, Parsa Pourali, and Dr. Ali Abedi. I value their valuable feedback and moral support. I would like to thank Mary McPherson for her outstanding help and treasured feedback on the writing of my thesis. I would like to express my utmost admiration and sincere gratitude to Dr. Amir Keyvan Khandani, for his great support.

Last but foremost, I want to thank my strong family because of their unconditional love and encouragement. I owe a lot of gratitude to my mother and my father who raised me with loving support for all my pursuits. Words cannot describe my thanks for faithful support my wife and my son provided me throughout my time at university. Their selfless sacrifice have let me exploit the opportunities that I would never have dreamed of. I also thank my brother and sister for helping to keep me motivated.

This work was financially supported by The Queen Elizabeth II Graduate Scholarships in Science and Technology, and Aapoolcoyuz Biman Memorial Graduate Scholarship in Engineering.

Dedication

This is dedicated to the ones I love.

Table of Contents

List of Tables

List of Figures

Abbreviations

AALL Array of Addresses' Linked Lists 42–45, 47, 76

ADLL Array of Distances' Linked Lists 42–45, 47, 76

ARI Adjusted Rand Index 50, 51, 65, 68, 74, 125–127

BALL Bidirectional Arrays of Linked Lists viii, 3, 13, 33, 35, 42, 48, 54, 76, 78–80

BIRCH Balanced Iterative Reducing and Clustering using Hierarchies 30, 31

CF Clustering Feature 31

CLARANS Clustering Large Applications based on RANdomized Search 30, 31

CURE Clustering Using REpresentatives 30, 31

DAC Discretized Agglomerative Clustering viii, 3, 13, 33, 35, 42, 46, 48, 53, 54, 56, 62–65, 67, 68, 70, 72, 74, 76–83, 105

DBDC Density-Based Distributed Clustering 29, 30

DBSCAN Density-Based Spatial Clustering of Applications with Noise 29, 30

DCS Distance Computation Step 9, 34, 36, 37, 39, 42, 45, 80

DIR Discretization Ratio 74

DPR Distance PReservation 112, 113

DR Dimensionality Reduction 105, 109–111

DUS Distance Updating Step 9, 34, 36–39, 42, 47, 60

FN False Negative 51, 124

FP False Positive 51, 124

HAC Hierarchical Agglomerative Clustering 8, 9, 12–14, 21, 23–26, 28, 30, 32–37, 39–42, 45, 47, 48, 52, 54, 57–65, 67, 68, 70, 72, 74, 76, 78–83

HC Hierarchical Clustering 21, 24, 25

LLE Locally-Linear Embedding 107, 113

LSH Locality Sensitive Hashing 115

MDS Multi-Dimensional Scaling 111, 112

MFS Minimum Finding Step 9, 34, 36–39, 42, 45, 47, 79

NLDR Non-Linear Dimensionality Reduction 112, 113

PAM Partitioning Around Medoids 30

PCA Principle Component Analysis 110–112

RI Rand Index 50, 51, 65, 70, 74, 125–127

ROCK RObust Clustering using linKs 25

SOM Self Organizing Map 25

SVD Singular Value Decomposition 111

TN True Negative 51, 124

TP True Positive 51, 124

TPR Topology PReservation 112, 113

Chapter 1

Introduction

The Lord said to Noah,"Go into the ark, you and your family and take with you a pair of every kind of animal, male and female, to keep their various kinds alive throughout the earth. There will be a flood which will wipe from the face of the earth every living creature." This very simple story provides the-first known application of large-scale cluster analysis.

In the modern world, several types of cluster analyses have been utilized in various fields such as statistics, genetics, software engineering, social sciences, and health-care. John Snow, the father of modern epidemiology, used cluster analysis to track the source of London's Cholera outbreak in 1854 [174]. In the information age, through technology improvements that have been facilitating data gathering, both the size of data and its dimensions have grown dramatically. Databases are among the examples in the field of computer science that have attracted cluster analysis as storing large data became an issue. The common idea of cluster analysis in all the examples above is to capture the association between data and to recognize groups within massive amounts of data.

In addition to clustering, other domains such as artificial intelligence, statistics, pattern recognition, machine learning, and knowledge discovery have intersections with data mining. Cluster analysis (or clustering) refers to the specific task of grouping a set of objects into subsets (clusters) based on their attributes (features) such that the similarity of objects in each cluster (intra-cluster similarity) is maximized and the similarity of objects between clusters (inter-cluster similarity) is minimized. Clustering reveals the hidden structure within data, but an exploratory technique and does not test hypotheses but actually generates them. These clustering descriptions are not precise, and it is not clear how to settle on one rigorous definition [166]. On the other hand, the lack of ground truth in unsupervised learning makes the evaluation of this exploratory task vague and even ambiguous.

Besides the fundamental difficulties of cluster analysis, another major problem may arise with growing data. When the size of data grows, either from an increasing number of objects or their dimensions, the memory and computational costs, as well as the quality of results, become a significant concern, are common to all data mining fields and similarly clustering. The nature and the amount of popular data in the Internet age threatens the scalability of many proposed clustering algorithms, even if the algorithms are widely used and reliably fit for small size and low dimensional data. The size of data grows either by the number of instances or each instance's dimension. When it comes to growing an instance's dimension, the phenomena of the curse of dimensionality and data sparsity arise: that is, when the dimension grows, the problem space grows so fast that the available data becomes sparse. This sparsity makes any machine learning or statistics task problematic, because finding patterns within data becomes difficult in two ways: First, more data is required to get reliable results because of the high-dimensional space where many dimensions are empty of meaningful data. Second, as searching for patterns within data may require inter-instance similarity computation, the computational cost grows linearly with the number of dimensions. Therefore, it is not only the number of instances that makes clustering (and most data-mining tasks) painful, but also the dimensions and variations in the nature of data, such as images, sounds, videos, and documents.

Clustering algorithms are not usually designed for a specific kind of data. The diverse nature of data (images, sounds, documents, etc.) usually does not change the clustering algorithms explicitly used for a specific kind of data. Originally, most clustering algorithms in the literature were explicitly designed by assuming that the objects in datasets are numerical data. Then, cluster analysis on other data types such as categorical or binary data began to use adjusted forms of the well-known algorithms designed for numerical data. In consequence, many such conventional algorithms are not fully adapted to other types of data. On the other hand, transferring from one space (possibly high dimensional) to a new space (possibly with lower dimensions) such as a binary space may help size-related challenges in cluster analysis algorithms.

Binary data is one well-known form and an interesting data type. Although there are applications of genuinely binary data, other types of data can also be represented by this compact yet meaningful version of data. The binary representation (codes or strings) of data may need less memory, and can effectively decrease storage and computational costs in many data mining disciplines. Thus, several approaches have been proposed in the literature for transferring data to compact binary codes, for various fields such as information theory, coding theory, statistical inference, and machine learning. One of the widely-studied approaches to this goal is *binary hashing*.

The goal of binary hashing is to transform data instances into a compact representation (low-dimensional) or a short binary code, also called a hash code. Hashing has two major branches: supervised [160, 161]–also called learning to hash–and unsupervised [36, 93]–also called locality sensitive hashing. Using compact binary codes that resulted from hashing helps in dealing with big data and its challenges, as it can decrease memory and potentially computational costs in data mining tasks. Memory costs decrease because the representative of each instance in a *target space* is a sequence of bits. The computational costs of using compact binary codes in many similarity-based data mining approaches can be decreased just by applying binary-specific distance/similarity measures, which are cheaper than other spaces.

All being said, one can decrease the costs of data mining in large size data by applying binary hashing on the *original space*, so as to get a compact binary representative of data instances. Then data mining approaches can be applied to the compact codes. Also, binary codes can potentially give us the freedom to modify data structures and algorithms and get rid of restrictions enforced by continuous numerical data. Accordingly, binary codes can be seen as opportunities to propose new data structures and algorithms to cut the costs of data mining tasks (clustering in this thesis).

In this thesis, *Discretized Agglomerative Clustering (DAC)-Bidirectional Arrays of Linked Lists (BALL)* is presented, a scalable data structure and algorithm that can be utilized to cluster binary data. This algorithm makes use of the nature of binary data and is based on a well-known clustering algorithm. Although the original algorithm is not efficient for large and high dimensional data, this solution efficiently fits binary data. Also, a data compression technique is used to find a new binary representation of data, and helps apply the proposed algorithm to continuous numerical data types.

Finally, the *efficiency* and *effectiveness* of the algorithm is evaluated on various datasets of different sizes and dimensionalities. Several clustering approaches and different compression ratios are explored to verify the quality of the algorithm and its scalability for various input data.

The clustering of binary data and its applications may benefit from the proposed approach and the techniques employed in this research. However, the work presented here is not complete, and several challenges remain in the field of binary data clustering. Therefore, possible future works are suggested at the end of this thesis.

1.1 Motivation

There was a time when oil companies were the most valuable corporations, and they controlled the world. However, now black gold is neither the most valuable asset nor the fuel for the future. The five most valuable public companies are Apple, Alphabet, Microsoft, Amazon, and Facebook by the total value of around 2.3 trillion dollars. Just after these huge corporations, comes Exxon Mobil to show that oil is still valuable but not as much as before. Hence, Intel CEO Brain Kraznich's declaration that "data is literally the new oil" is not far from reality.

As of April 2017 the number of Internet users had grown to 3.8 billions ([4]). Every day, 2.5 quintillions more bytes of data are created. Ninety percent of data in the world today has been created since 2016 ([1]), and the growth rate will increase drastically with new devices and technologies that are now becoming connected as a part of Internet of things applications. A straightforward example of the flood of data is in autonomous vehicles. Every autonomous vehicle produces 4 Tera-bytes each day with its cameras, radar, sonar, and GPS. According to the Barclays report [2], it is expected that in the next decade each car will produce 100 gigabytes of data every second. Thus, the US alone, with its 260 million cars, would produce 5.8 Zetta-byte of data each day. This amount of data is peanuts compared to the data that will be generated by all the connected devices on the Internet of things (IoT), from home appliances to human-health monitoring systems. "Big data" is another term for designating large-scale data that has several million or even billions of instances.

This huge flood of data is not researchers, and data scientists, worst fear; the dimension of data and the wide variety of data types for each dimension are more threatening to conventional data science approaches. Having powerful machines cannot help deal with such a variety of data, all by itself. Accordingly, in the literature, there are a surprising number of approaches proposed to cope with the diversity of data. To assay these data mining and machine learning algorithms, data scientists have produced datasets with various dimensions and data types. From the long list of large, high-dimensional datasets, Imagenet [57], TexMex [97], Youtube-8M [6] can be mentioned; all of which can be used for assessing various data science tasks. These tasks include cluster analysis, classification, machine vision, data visualization, or in summary, information extraction and knowledge discovery.

Regardless of the knowledge discovery task, not all machines can handle enormous amounts of high-dimensional data. On the other hand, all machine learning processes cannot be taken to powerful servers, and some machine learning tasks should be handled on local machines. Autonomous vehicles and smart portable devices are two examples where performing data mining tasks independently using local machine is crucial. Some machines have limitations on memory, and some have vital responsibilities where computational time is critical. An autonomous vehicle has neither a built-in huge and powerful server and nor a connection to servers. Also, the vehicle has to decide its subsequent actions within milliseconds as soon as surrounding conditions change. These examples make clear the threat of big data to any knowledge discovery task. In this thesis, the focus is on a specific segment of knowledge discovery: clustering. Although clustering is a part of knowledge discovery, data mining or machine learning, finding an algorithm that can deal with big-data in high-dimensional space may open doors for investigating other fields, such as classification.

Clustering of a large-scale data can be applied to many tasks, including clustering the products in an online store such as books on Amazon, images on a social media network such as the billions of images on Instagram or Facebook, web pages based on their content, and documents such as tweets on Twitter. The goals of these applications range from marketing (i.e., similar product recommendations) to finding duplicate content, speeding up search queries, or detecting cyber-attacks. It is interesting that clustering all by itself can help in coping with the flood of data. Therefore, clustering can affect big data or be affected by it.

Numerous studies have addressed large-scale data on various data mining and knowledge discovery domains, such as pattern recognition, machine learning, statistics, clustering and related segments and their application in different fields of science. As mentioned, cluster analysis is an ill-defined problem and shows its challenges more in large-scale high-dimensional data. Fortunately, the necessity of finding ways to face large-scale data clustering has forced researchers to propose various approaches. Regarding high-dimensionality, one can select either of two primary directions, *preprocessing* such as data compression techniques before clustering, or *clustering high dimensional data* such as subspace clustering to deal with the high dimensionality of data. Also, three primary approaches are available to cope with the huge number of instances: first, reducing the number of passes over the data; second, accessing a portion of data (not whole instances); and third, parallelizing computation. Obviously, none of the proposed algorithms are suitable for all datasets and fit all clustering problems. For instance, stream data clustering is one crucial example of big data. The settings of clustering problems for batches and streams are different, so it would be challenging to make any proposed approach scalable for both batch and stream data clustering.

All being said, the need to improve the speed and quality of information retrieval and machine learning algorithms has attracted researchers' interests. There is also a lack of an exact deterministic clustering algorithm that has acceptable performance for high-dimensional big data. Thus, in this context, the performance and quality of deterministic clustering algorithms still remain the issue.

1.2 Problem Description

The clustering problem is one of the oldest statistical problems and has been studied widely in the literature. It can be defined as follows:

Definition 1. *Given a set of records,* ***clustering*** *is to find a function that partitions records into a set of groups which are as similar as possible .*

Several clustering algorithms can be derived and categorized from this definition and based on the definition of similarity that will be covered in the next chapter. As can be seen from the definition, the core idea in clustering is to find the closest (i.e., the most similar) data points and put them in a cluster. Therefore, in many algorithms, clustering can be viewed as a nearest-neighbour search problem. In addition to similarity measures that change the clustering problem, the data type also affects the definition of the clustering problem. Similarity measures can also profoundly affect the speed of the clustering. Therefore, the data type or the clustering space can potentially change the solution speed.

Many approaches have been proposed to improve the accuracy and speed of solutions to the problem. Large-scale datasets justify the importance of fast and accurate clustering approaches. The problem can be divided into, first, finding the space that could make similarity/distance calculation fast, and second, devising an algorithm that speeds up the process of nearest neighbour searching as much as possible. The solution to the first part of the problem can affect many clustering approaches, mostly in the distance-based category. By transferring the original space into a target space where similarity measure computation is fast, one achieves faster cluster analysis in many cases. On the other hand, one effective requirement for efficient clustering is to speed up the process of the nearest-neighbour search. Several exact and approximate solutions have been proposed to satisfy this requirement. As many clustering approaches use nearest-neighbour searching, one should attack a specific clustering algorithm for the second aspect of the problem. This research is mainly focused on the second aspect, where the goal is to find a data structure and algorithm that speeds up the pairwise nearest neighbour search, and consequently makes hierarchical clustering feasible in large binary datasets. In order to expand the advantage

Table 1.1: Table of Notation in This Chapter

a, b	$\triangleq$	data-points such that $a \in A$ and $b \in \mathcal{B}$
$\mathbf{b}_i$, $\mathbf{b}_j$	$\triangleq$	Binary Code vectors
C_m	$\triangleq$	Cluster m
d	$\triangleq$	Number of dimensions in binary space
$dis_{ij} \equiv dis(i, j)$	$\triangleq$	Distance between two clusters or points i,j
D	$\triangleq$	Number of dimensions in Real space
n	$\triangleq$	Number of instances
$\mathbf{p}_1 ... \mathbf{p}_n$	$\triangleq$	Data-points as clustering inputs
$\mathbb{R}^D$	$\triangleq$	Real space with D dimensions
$Sim(i, j)$	$\triangleq$	Similarity between two clusters or points i,j

of the proposed algorithm on different types of data, well-known hashing techniques for mapping data to binary space have also been utilized.

Based on the motivations above and the problem description, the objectives of this research are to:

- Carry out comprehensive study of current techniques and strategies for data compression such that a single machine can handle the compact data.

- Design and develop a new data structure and algorithm that speeds up the clustering of binary data without losing the quality of clustering.

- Evaluate and compare the proposed algorithm to existing clustering approaches.

Table 1.1 shows the notation used in the thesis.

1.3 Definitions, Research Scope And Problem Statement

Definition 2. *Given a set of records, $\mathcal{B}$, with n items, where $\mathcal{B} = \{\mathbf{b_1} ... \mathbf{b_n}\}$, $\mathbf{b_i} \in \{0, 1\}^d$ and $i \in \{1...n\}$, and a function, dis, which is responsible for finding the distance between* *items, **the clustering** is done to find a function, f, which gives a partition of $\mathcal{B}$.*

Definition 3. *A **hierarchical clustering** S on a set of records $\mathcal{B} = \{\boldsymbol{b}_i\}_{i=1}^{n}$, $\mathbf{b_i} \in \{0,1\}^d$ is a set of clusters such that:*

(i) $\exists C_0 \in S | C_0 = \mathcal{B}$

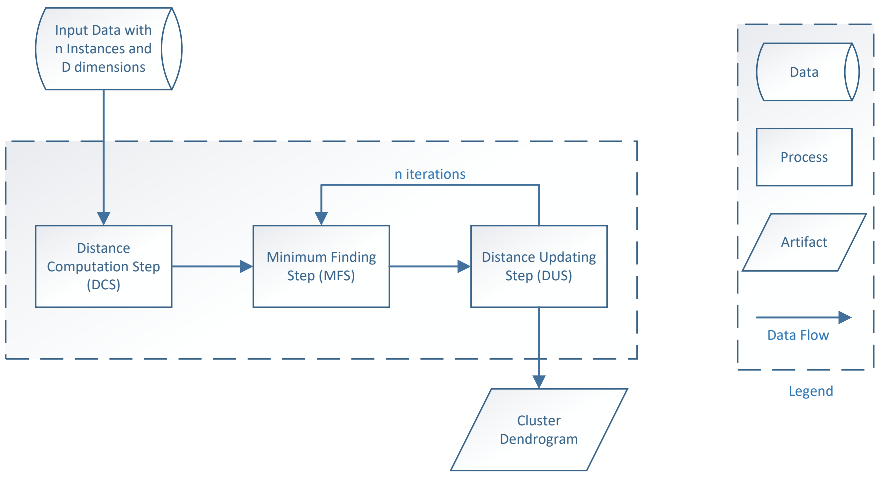
(ii) $\forall \boldsymbol{b} \in \mathcal{B}, \exists \{\boldsymbol{b}\} \in S$

(iii) $\forall\ C_i,\ C_j \in S:\ \ (C_i \subset C_j)\ \ \vee\ \ (C_j \subset C_i)\ \ \vee\ \ (C_i \cap C_j = \emptyset)$

(iv) $(\forall C_i \in S) \wedge (C_i \neq \emptyset)\ \ , \exists \{C_j\}_{j=1}^{k}\ \mid\ (C_j \in S) \wedge\ \ (\forall C_m, C_n : C_m \neq C_n,$
$C_m \cap C_n = \emptyset)\ \ \wedge\ \ \cup_{j=1}^{k} C_j = C_i$

Definition 2 is the general definition of clustering in binary space. Such clustering is the superset of hierarchical agglomerative clustering (i.e., definition 3). These definitions do not talk about the process of clustering or HAC. The graphical representation of the HAC algorithm is shown in Figure 1.1 to give a broad overview of the HAC process. HAC is merely the inter-instant similarity computation, and then $n - 1$ merging steps that are executed, each of which combines the two most similar clusters (or singletons). After mathematically defining the HAC and presenting an overview of its process, it is essential to clarify the scope of this research formally. This research will explore approaches to speeding up the HAC problem, summarized as follows [11, 173]:

- *Agglomerative*: the algorithm starts from n singleton clusters, and groups them until it arrives at one single cluster that contains all n instances.
- *Hierarchical*: it creates a dendrogram of clusters that presents the partitioning sequences from n to one cluster.
- *Non-overlapping*: in it, each partition is a disjoint set at each level of the dendrogram (i.e., the sequence of partitions).
- *Sequential pair-group*: it iteratively agglomerates clusters such that two clusters are merged at each iteration.
- *Provided information*: two forms of information are usually provided: first, making a *stored matrix* approach that includes non-negative values of inter-instance dissimilarities, and second, making a *stored data* approach that stores objects ($\mathbf{x}$) by a D-tuple of real or binary numbers. D can be considered as the number of dimensions, and x_i is the amount pertaining to the ith dimension.



Input Data with n Instances and D dimensions
n iterations
Distance Computation Step (DCS)
Minimum Finding Step (MFS)
Distance Updating Step (DUS)
Cluster Dendrogram
Data
Process
Artifact
Data Flow
Legend

Figure 1.1: Overview of Agglomerative Clustering Process

The scope of this work is the *hierarchical*, *agglomerative*, *non-overlapping*, *sequential pair-group* aspects of *exact* clustering algorithms, where the input is *binary stored objects*.

Problem Statement

A scalable algorithm is needed to promote Hierarchical Agglomerative Clustering (HAC) with full coverage of linkage criteria and good practical performance.

Now that the scope of this research has been introduced, it is time to shed light on the primary issue with the algorithm. The HAC algorithm (Figure 1.1) is a summary of the steps in HAC. Figure 1.1 shows the four simple steps of HAC:

1. *Distance Computation Step (DCS)*: In this step, the algorithm computes the inter-instance distances and store them. The computational complexity of this step is $O(n^2D)$.

2. *Minimum Finding Step (MFS)* or Pairwise Nearest Neighbor Search: In order to merge two clusters, the closest pairs should be found, which is the responsibility of this step and has the computational complexity of $O(n^2)$.

3. *Distance Updating Step (DUS)*: After two clusters are merged, the algorithm needs to update the distance between this newly merged cluster to all other clusters ($O(n)$). Computing of the updated distances is performed in this step.

4. Iterate through steps two and three until only one cluster remains.

The algorithm iterates through step two and three n times. If $D << n$, the computational complexity of this algorithm becomes $O(n^3)$. The space complexity of the algorithm is $O(n^2)$. The costly nature of this algorithm is hugely dependent on the number of instances and the number of dimensions.

The following section (1.4) discusses challenges to be considered in working toward a comprehensive solution.

1.4 Research Challenges

Some challenges must be dealt with in finding a solution to the problem:

Costly size dependency: The computational cost of a clustering algorithm must be tolerable. Most clustering algorithms work based on the similarity/dissimilarity of items in a dataset. Therefore, the computational cost of a clustering algorithm is proportional to the number of comparisons times the cost of each comparison. The cost of distance computation is discussed under the next research challenges (memory costs and dimensional dependency), but briefly, as the size of a dataset expands, the number of comparisons also grows, and the growth rate depends on which clustering algorithm is used. Here, the goal is to find solutions for the pairwise nearest neighbour search of a massive dataset, which requires a quadratic time algorithm. Thus, performing this search multiple times makes clustering a cubic algorithm in some classes of algorithms, which is the primary challenge here.

Space complexity of storing and cluster analysis: Moreover, as the size of datasets grows, solutions must have adequate space complexity. The ideal condition for data representatives is to fit into the working memory of a single machine. Although this ideal condition cannot happen for massive datasets, compressing data as much as possible without losing its semantics is still a goal. In this research, the aim is to cluster such a compressed version of data. It is necessary to speed up the clustering without compromising the space complexity and the clustering accuracy. This complexity preservation is the second challenge in this research.

Dimensional dependency of similarity computation: Memory costs have motivated an exploration of compact-data-specific clustering. Additionally, a clustering solution needs to compute distances. The cost of distance computation grows linearly with the number of dimensions, which in high dimensional datasets may become an issue. This cost should be tolerable, so another challenge has been to optimize a function that decreases the computational complexity of distance computation by lowering dimensional dependency.

1.5 Research Questions

The following research questions have been posed to deal with the memory and computational cost challenges discussed in the previous section:

RQ1: Given a compact discrete version of the data or its inter-instant distances, is it possible to develop a data structure and algorithm in a way that minimizes the pairwise nearest neighbour searches of a hierarchical agglomerative clustering algorithm?

It is evident that such a question is highly dependent on the data type of compact code. Proposing a data structure and algorithm that performs efficiently on a data type depends on the nature of the data. On the other hand, developing a compact version of the data that does not help decrease the computational cost of nearest neighbour searching is ineffectual.

Several solutions have been introduced to decrease the dimensionality of data (or compress the data), so as to improve the efficiency of distance computation, but another question remains: is there any possibility of performing a nearest-neighbour search at each clustering step with less computational cost on compact codes? The goal is a solution that can remove the need for exhaustive searches for the closest clusters by using a data structure that provides pairwise nearest neighbour searching with close-to-constant computational complexity. This solution significantly reduces the computational cost of agglomerative clustering, as finding the pairwise nearest neighbours is the dominant part. Therefore, if no data structure has been specially designed to get rid of exhaustive searches, this problem would be almost as costly (even on compact codes) as ordinary agglomerative clustering or finding nearest neighbours n times. Thus, the primary goal is to design a data structure and algorithm that can improve clustering times by using compact codes (binary codes). So the next research question is:

RQ2: If there is such a data structure and algorithm, is there any specification for the type of data?

Suppose that finding a data structure and algorithm that can decrease the computational complexity from cubic to quadratic is achievable. One concern would be the *space complexity* of that solution. A reason behind using compact codes is to ensure that the data fits in the memory of a single machine in order to prevent transferring back and forth between memory and storage devices. It is thus crucial to know how much memory, in the worst case, is needed at any point in the algorithm (i.e., space complexity). As with time complexity, the researchers are mostly concerned with how the space grows as the size of the input problem grows. This necessity leads to the third research question:

RQ3: Which data structure can be used so that the space complexity of hierarchical agglomerative clustering does not increase prohibitively?

These research questions will be answered in Chapter 4.

1.6 Thesis Contributions

Research has addressed big data in various data mining and knowledge discovery fields such as pattern recognition, machine learning, statistics and related segments, and their application in different fields of science. However, none of the proposed algorithms are suitable for and fit all clustering problems on various datasets. Each clustering algorithm has advantages and disadvantages. Many fast clustering algorithms are not deterministic, which means the solution's starting point affects the results. In some, prior knowledge, such as the number of clusters, is essential. The family of *Hierarchical Agglomerative Clustering (HAC)* algorithms involves a group of algorithms that are deterministic and no prior knowledge of the number of clusters is needed. However, they are computationally expensive, making this family unsuitable for large-scale data. The popularity and power of these algorithms and, at the same time, the simplicity of working with binary codes and vigorous hashing algorithms is a motivation to consider combining these two and to propose a data structure and algorithm. The proposed approach brings the agglomerative algorithm close to its lower bound. This thesis introduces an algorithm that is a combination of the data structure and algorithm, and binary hashing is a patch to this algorithm.

The following are the contributions of this thesis.

- Discrete distances are adapted to enable HAC to perform the following:
  - It merges clusters (points) considering closest pairs without a pairwise nearest neighbour search.
  - It updates all corresponding distances while merging two clusters without violating the consistency of stored sorted distances.
  - It enables random access to the distance between two clusters.

- A combination of data structures is introduced (DAC-BALL) that performs the following:
  - It enables DAC to execute exact HAC on binary data with the lowest computational complexity that the researchers are aware of.
  - It enables DAC to perform exact HAC on any bounded discrete space by the lowest computational cost.
  - Transferring $\mathbb{R}^D$ to $\{0,1\}^d$ enables efficient approximate HAC in $\mathbb{R}^D$ with small accuracy loss, and is drastically faster than the conventional HAC.
  - It creates the possibility of approximate HAC in $\mathbb{R}^D$ by discretizing the space such that distances are finite integers. This approach needs no transformation to $\{0,1\}^d$.

The novel clustering algorithm proposed in this dissertation takes advantage of binary codes. Due to the high computational costs of conventional HACs, utilizing them on large-scale datasets is not feasible. Therefore, having scalable agglomerative clustering approaches, such as the proposed algorithm, is necessary. Although the algorithm is explicitly designed for binary space, the proposed approach becomes applicable for any type of data by patching the algorithm with binary hashing. Even though the lower bound of hierarchical clustering can be higher than that of some flat clustering algorithms, the deterministic and prior-knowledge-free properties of this family of algorithms are valuable.

1.7 Thesis Organization

This thesis contains five chapters. Chapter 2 surveys pertinent literature on the problem of clustering for categorical data and compact codes. Chapter 3 presents the foundation of the discretized agglomerative clustering (i.e., DAC) algorithm and corresponding data structure (i.e., BALL). Chapter 4 presents the results for clustering the real space and binary space of various datasets. Chapter 5 wraps up the thesis and suggests some directions for future research.

Chapter 2

Background Concept and Related Work

In this chapter, the related work in the domain of clustering is reviewed. Because this research attempts to use the compact versions of data, it seems reasonable to first review some dimensionality reduction approaches here in this chapter. However, the aim of this research is not to develop new dimensionality reduction and data compression techniques. Briefly, the goal is to develop a hierarchical agglomerative clustering algorithm that uses the discrete, bounded nature of binary data as an input for clustering. This binary input data can be real-world binary data or the result of a dimensionality reduction or data compression technique such as binary hashing. Therefore, a dimensionality reduction or data compression technique is only a patch to the proposed approach. To cover the idea, a brief overview of these techniques is provided in Appendix A.

In this chapter, some clustering algorithms are briefly reviewed. The studies on clustering are too numerous for an in-depth treatment here. They can be narrowed down to the ones most related to this research: hierarchical agglomerative clustering techniques and their optimized approaches. However, as one of the goals of optimizing HAC algorithms is to enable applying them to big data, some algorithms that apply to big-data clustering are also described here. Agglomerative clustering algorithms accept various types of data such as continuous numerical and categorical types. In this thesis, the research is concerned with a particular setting in which the input instances are represented with binary strings, which are a subset of categorical data. Accordingly, some categorical data and hierarchical clustering algorithms are described.

There is an assumption worth mentioning here: the distance metrics for input data are already defined. As many clustering algorithms are based on distance computation, having defined distance metrics is crucial in this domain. Although there are several active research venues on learning distance metrics, it is assumed that these metrics are already defined for binary strings.

While several approaches for clustering exist, the discussion here is focused mostly on topics related to big data, in particular, high dimensional, large-scale, discretized bounded data.

In this thesis, American mathematical typography is followed. Italic type is employed for all letters representing variables and parameters. Bold Latin capital letters usually represent vectors, and bold lowercase letters are used for vectors. The names of well-known functions are written in lowercase upright letters.

2.1 Data Clustering

Clustering means grouping data. It has a long history in the human brain and in the brains of many creatures from chimpanzees to crows. Many clustering examples exist, from flat clustering (e.g., Noah's approach for clustering creatures) to hierarchical clustering (e.g., Aristotle's pattern recognition for living things). The field is still an active research area in the literature; hundreds of thousands of clustering-related documents have been published. In the following sections, a simple, systematic overview of clustering is provided, but the goal is not to dig into mathematical details for the various cluster analysis methods as that would lead to a very long book.

Given a set of items, a desired number of clusters (optional) and an objective function, the goal of clustering is to assign cluster labels to the items so as to minimize the objective function. The "objective function" can be merely the distance between items. Clustering is a problem of finding the best assignments that minimizes an objective function. Clusters do not possess a unique definition, and no one can define the best clustering approach on an abstract dataset. Thus, cluster analysis is highly context dependent. A user needs to find the best clustering approach (within the existing paradigm) that is the optimum for her needs. Next, some widely-used terms in the cluster analysis field are clarified.

Flat vs *Hierarchical*: If each cluster (super-cluster) is partitioned into sub-clusters, then cluster analysis is hierarchical; otherwise, the clustering is flat.

Hard vs *Soft*: If each object belongs to only one cluster, it is called hard clustering. In contrast, if each object has a degree of membership (probability distribution) to clusters, it is called soft clustering.

Exhaustive vs *Exclusive*: If one partitions the data such that every object belongs to a cluster and no object is left without assignment, the clustering would be exhaustive. Contrarily, if the method allows some objects (e.g., outliers) to not be assigned to any cluster, the clustering is exclusive.

Probabilistic vs *Non-probabilistic*: If a clustering method is based on probabilistic models (e.g., finite mixture approaches), it is called probabilistic; otherwise, approaches without probabilistic interpretation that are based on minimizing a cost are called cost-based or non-probabilistic.

Other terms such as *parametric* vs *nonparametric* (i.e., based on required input parameter), or *partitional* vs *partitional hierarchical*, or the *cardinality* of clustering (the number of clusters) are less-common terms.

2.2 Cluster Analysis Classification

Cluster analysis has acquired many applications and approaches over several decades. Here, clustering is reviewed according to its algorithmic approach and methodological viewpoint during the past decades. Several other ways exist to classify cluster analysis methods. The aim is not to review the extensive literature on cluster analysis, but only limited to the most-influential methods and applications in this field. The taxonomy of clustering approaches will be shown in tree representations to give a better overview. It is worth mentioning again that some of the clustering approaches can fit into multiple classes, and other variations of this classification are also viable. One simple example is the k-means algorithm, which can be both a partitioning-based algorithm (in the current taxonomy) and a centroid-based (other taxonomies) one.

The following taxonomy of clustering is focused on categorizing based on the approaches used, not the type or size of data. Hierarchical, partitioning, density-based, spectral-based, and model-based techniques have been put forward mainly to solve the clustering problem from different viewpoints without too much attention to data-type. Different applications may have different data-types such as categorical data or network data. Some clustering techniques work better on some data-types, and tuning can even sometimes be applied to a technique to comply with a specific data-type. In this section, clustering methods are classified by looking only at techniques and not the size of data or data-type. The big-data and categorical type clusterings are discussed, as they are close to the current problem covered in subsequent sections.

In Figure 2.1, clustering algorithms are classified into five branches. In some cases, such as hierarchical clustering algorithms (as an example), the branches have subtrees that in this case are the *Conventional* and the *Modified* hierarchical clustering algorithms. One of the most important performance measures for an algorithm (efficiency measures) is the computational complexity. The computational complexity of algorithms in Table 2.1 shows a comparison between the speed of these clustering algorithms. Some of these algorithms have multiple variations, so the computational complexity of the original versions are shown (alongside their references). Other parameters such as the number of input parameters, handling large datasets, handling high dimensionality, handling noisy data, type of dataset, deterministic or non-deterministic, exact or approximate, and cluster shapes are not compared here, because these are outside the scope of this chapter.

Here, the clustering techniques are categorized into five branches, but other forms of taxonomy are also possible. Even in this taxonomy for instance, one can change the shape of the tree by putting many algorithms into *distance-based* algorithms. Several ways of sub-categorizing distance-based algorithms exist, but one simple way is to put algorithms into two subsets: hierarchical and flat techniques. Hierarchical methods build a hierarchy of clusters with varying levels of granularity. Strategies for hierarchical clustering generally fall into two groups: (i) *agglomerative*, which builds a bottom-up hierarchy from singleton clusters to one cluster, and (ii) *divisive*, which builds a top-down hierarchy from one cluster to singleton clusters. Agglomerative clustering methods can be categorized into *linkage-base* and *graph-based*.

The examples of agglomerative, divisive, linkage-based, and graph-based algorithms can be found in Figure 2.1. K-means, k-medoids, k-medians are examples of flat clustering.

2.3 Clustering Categorical Data

Cluster analysis is performed on two types of records: first, records with real-value continuous features (attributes), and second, records with categorical (discrete) features. The main focus of most studies in clustering is on real-value continuous data clustering. One major example is clustering images in which the data is of a continuous domain. In contrast, clustering in categorical (discrete) space is a newer problem. Partitioning large software systems and protein interaction data are two applications for non-ordered (since there is no ordering for features values in this problem) discrete data clustering.

Binary data is the simplest form of categorical data, in which features may take one of two possible values (from $\{0, 1\}$). Binary datasets are often sparse and sometimes can have any number of features (e.g., market basket clustering). Although clustering

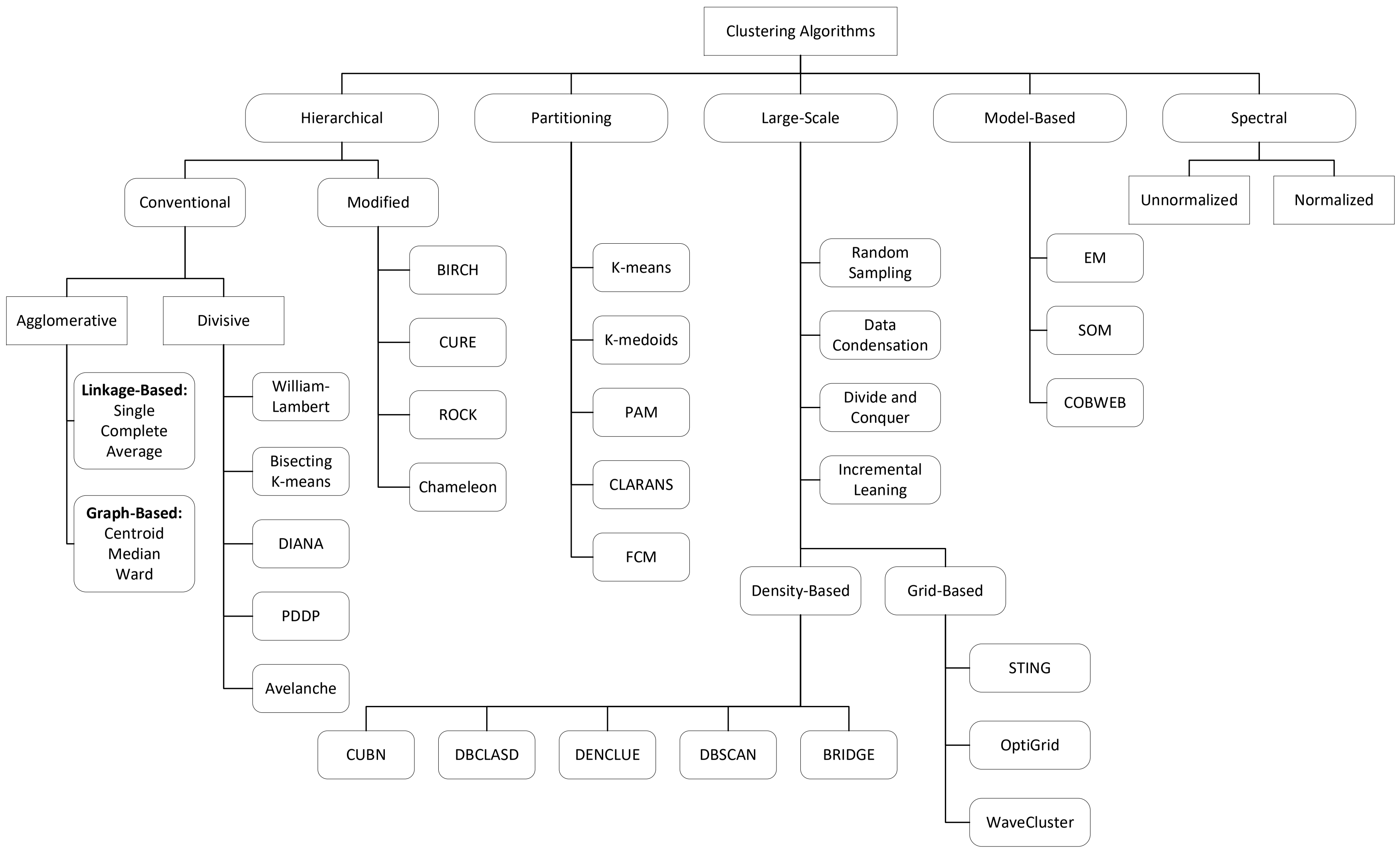


Figure 2.1: Taxonomy of Clustering Algorithms

Category	Algorithm	Computational Complexity	Input Parameter	Advantages	Disadvantages
Hierarchical	Agglomerative [133, 106, 173, 190, 197]	$O(n^3)$	No	Arbitrary shapes; Interpretable	Relatively costly
	Bisecting K-Means [176]	$O(ktn)$	Yes		
	DIANA [104]	$O(n^4)$	No		
	PDDP [22]	$O(K_{max}.(2+t_{SVD}).S_{nZ}.n.d)$	Yes		
	Avalanche [9]	$O(n^3)$	No		
	ROCK [82]	$O(kn^2)$	Yes		
	Chameleon [103]	$O(n^2)$	Yes		
	LIMBO [14]	$O(n\log(n))$	Yes		
	BIRCH [204]	$O(n)$	Yes		
	CURE [81]	$O(n^2\log(n))$	Yes		
Partitioning	K-Means [130]	$O(tnkd)$	Yes	Relatively efficient	Sensitive to noise; Just convex shapes; Local optima
	K-Medoids [104]	$O(tn^2d)$	Yes		
	PAM [104]	$O((n-k)^2k)$	Yes		
	CLARANS [142]	$O(n^2k)$	Yes		
	FCM [21, 64]	$O(n)$	Yes		
Density-Based	DBSCAN [66]	$O(n^2)$ $(O(n\log(n))$ special index)	Yes	Relatively efficient; Arbitrary shapes	Low quality; Memory usage; Parameter sensitive
	DENCLUE [89]	$O(n^2)$	No		
	DBCLASD [200]	$O(n^2)$	No		
	BRIDGE [50]	$O(tnkd+n\log(n))$ special index	Yes		
	CUBN [188]	$O(n)$	Yes		
Grid-Based	STING [189]	$O(k)$ (k: No. cells)	Yes	High efficiency; High quality	Mesh size sensitive
	OptiGrid [88]	$O(nd)$	Yes		
	WaveCluster [168]	$O(n)$	Yes		
Model-Based	EM [56]	$O(knp)$	Yes	Interpretable	Relatively costly; Parameter sensitive
	SOM [108]	$O(Mn^2)$	Yes		
	COBWEB [69]	$O(n^2)$	No		
Spectral	Normalized Spectral [169, 141]	$O(n^3)$	No	Arbitrary shapes	Relatively costly
	Unnormalized Spectral [185]	$O(n^3)$	No		

Table 2.1: Broad Overview of Various Clustering Algorithms

in continuous space has been widely studied and several powerful algorithms have been proposed, clustering algorithms for categorical data (specifically binary data) should be optimized based on the discrete nature and specific properties of discrete data.

The definition of categorical clustering is quite similar to the definition of continuous space clustering. Clustering partitions n objects into k clusters. Each object, $\mathbf{O}_i$, has d attributes ($\{O_{i1}, O_{i2}, ...O_{id}\}$). Each attribute, O_{ij}, has a domain. This domain for real-value space involves real values and otherwise takes a categorical or Boolean data-type. The previous classification approach (clustering taxonomy, 2.1) is followed here to classify the major categorical clustering algorithms.

Here, similar taxonomy is used, but specifically for categorical data. Although many of the clustering algorithms in Figure 2.1 can be applied to categorical data, some algorithms are suitable or tuned for this data-type. Table 2.2 shows these algorithm with their computational complexity.

Other parameters, such as the number of input parameters; the handling of large datasets, high dimensionality, and noisy data; the type of dataset; deterministic or non-deterministic; exact or approximate; and cluster shapes are not compared here, because these are outside the scope of this chapter.

Category	**Clustering Algorithm**	**Computational Complexity**
Hierarchical	ROCK [82]	$O(kn^2)$
	Chameleon [103]	$O(n^2)$
	LIMBO [14]	$O(n\log(n))$
	COBWEB [69]	$O(nd^2)$
Partitioning	K-Medoids [104]	$O(tk(n-k)^2)$
	K-Modes [90]	$O(tkn)$
	CLARA [104]	$O(ks^2+k(n-k))$
	CLARANS [142]	$O(n^2)$
	Squeezer [86]	$O(kn)$
Density-Based	OPTICS [15]	$O(n\log(n))$
	CLOPE [201]	$O(kdn)$
	HIERDENC [13]	$O(n)$
Model-Based	AutoClass [37]	$O(tkd^2n)$
	SVM [196]	$O(n^{1.8})$

Table 2.2: Complexity of Various Categorical Clustering Algorithms

2.4 Hierarchical Clustering Algorithms

One of the most widely used families of algorithms for cluster analysis is hierarchical clustering. The simplicity of these algorithms and ease of interpreting their results widen their application to various fields of science.

Hierarchical Clustering (HC) algorithms attempt to solve problems by making a dendrogram. The dendrogram is merely a binary tree constructed by merging two clusters or splitting a cluster. One interesting fact about hierarchical algorithms is that, in contrast to some other approaches, they do not need a predefined number of clusters. Accordingly, different levels of the dendrogram represent different numbers of clusters. By this means, the dendrogram holds all clustering results from one to n clusters. In contrast, for the k-means as an example for two different ks, one must run the algorithms twice.

Two branches of the family of HC algorithms are top-bottom (i.e., divisive) and bottom-up (agglomerative), but the concept of closeness –similarity or dissimilarity (distance)– is common between them.

In the rest of this section, HC algorithms and their specifications are briefly discussed because the proposed approach is based on this type of algorithm.

Agglomerative Approaches

There are two branches of HC algorithms: agglomerative and divisive. Agglomerative algorithms start off by considering all instances as singleton clusters and iteratively merge them based on their closeness until only one cluster remains. The result is a dendrogram with singleton instances in the leaves and entire data instances in its root. The dendrogram is common in agglomerative and divisive algorithms as the leaves of this tree are singleton clusters, and the root is a big cluster that includes all instances. However, the dendrogram structures in agglomerative and divisive approaches may be entirely different.

Different similarity/dissimilarity is the key to different versions of agglomerative algorithms, but apart from this, all agglomerative algorithms follow common steps: *building a distance matrix*, *merging the two closest clusters* (i.e., at first, all data points are considered as a singleton cluster), *updating the distance matrix*, and doing step 2 and 3 until there is only one cluster.

The closeness definition determines the different versions of agglomerative algorithms:

Single-Linkage: One of the oldest Hierarchical Agglomerative Clustering (HAC) algorithm is single-linkage clustering [133]. The similarity between two sets of clusters, A and B, of the single-linkage algorithm is defined by:

$$d_{AB} = \min\{dis(a,b) | \forall a \in A, \forall b \in B\} \tag{2.1}$$

This algorithm is also known as nearest-neighbour clustering as the similarity between two clusters is the similarity between two nearest neighbours in these two clusters. Because of this property, the single-linkage algorithm can find asymmetrical cluster shapes. However, this similarity makes single-linkage easily compromised by the noise in the data, and the chaining effect is another major drawback (Figure 2.2). Since the merging criterion is local, a chain of points can be merged regardless of the overall shape of clusters.

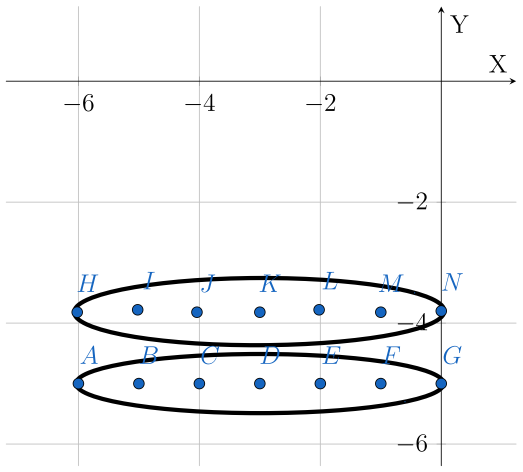


Figure 2.2: Chaining Effect in Single-Linkage HAC

Complete-Linkage: In contrast to the single-linkage approach, it has been proposed that the proximity measure between two clusters is the most dissimilar data point of those clusters. This approach is called complete-linkage [106]. Thus, dissimilarity between two sets of clusters A and B of the complete-linkage algorithm is defined thus:

$$sim_{AB} = \max\{dis(a,b) | \forall a \in A, \forall b \in B\} \tag{2.2}$$

This approach eliminates the chaining effect of single-linkage, but it is more sensitive to noise and outliers than single-linkage, as can be seen in Figure 2.3. Point A in this figure is an outlier. Although the distance between B and C, D, E is more than the distance between B and A, so clusters A and B merge together.

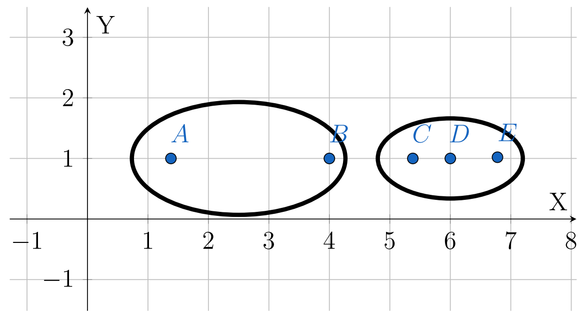


Figure 2.3: Outlier Sensitivity in Complete-Linkage HAC

These proximity differences define the distance between two clusters and only change the update step of the HAC.

Group Averaged-Linkage: Two approaches have been proposed to eliminate the drawbacks of single and complete-linkage algorithms. One of these approaches is group averaged-linkage or simply average-linkage, which defines the distance between two clusters A, and B, as the average distance between all pairs of points between these two clusters:

$$sim_{AB} = \frac{1}{|A|.|B|} \sum_{a \in A} \sum_{b \in B} d(a, b) \tag{2.3}$$

This approach merges two clusters with the highest cohesion [132]. Another approach for eliminating single and complete-linkage drawbacks is *centroid-based* HAC. In this approach, the distance between two clusters is defined as the distance between their centroids.

$$sim_{AB} = \{d(\mathbf{c}_A, \mathbf{c}_B) | \forall a \in A, \mathbf{c}_A = \frac{1}{|A|} \sum_{a \in A} \mathbf{a}, , \mathbf{c}_B = \frac{1}{|B|} \sum_{b \in B} \mathbf{b}\} \tag{2.4}$$

These two algorithms–specifically group average-linkage–are computationally more expensive than the single and complete-linkage algorithms.

Ward [190] or *minimum variance* clustering algorithm is another approach in HAC. If $\mathbf{m}_A$ is considered as the mean or centroid of cluster A, then:

$$\mathbf{m}_A = \frac{1}{|A|} \sum_{\forall \mathbf{a} \in A} \mathbf{a} \tag{2.5}$$

and therefore, the variance of a cluster is:

$$\mathbf{s}_A = \sum_{\forall \mathbf{a} \in A} (\mathbf{a} - \mathbf{m}_A)^2 \tag{2.6}$$

In this approach two clusters would be merged if their newly merged cluster has the minimum variance (i.e., $\mathbf{s}_{AB}$) between all other possible merging options. Ward method has a very close relationship with the centroid method, as ward proximity is calculated by:

$$w_{AB} = \frac{|A|.|B|}{|A| + |B|} d(\mathbf{c}_A, \mathbf{c}_B) \tag{2.7}$$

where $d(\mathbf{c}_A, \mathbf{c}_B)$ is the distance between the centroids of two clusters, A and B, weighted by a product of the size of these clusters.

Divisive Approaches

Another branch of hierarchical algorithms is the top-down (or divisive) approaches. The focus in this thesis is on HAC, but to cover the whole family of HC, divisive algorithms, major points are briefly discussed.

Divisive algorithms start by considering all data points in a big cluster, and then try to iteratively split this cluster and newly formed clusters based on similarity/dissimilarity and build the dendrogram. Assume that hundreds of thousands of data points are available and the goal is to cluster them into a few numbers of clusters (k). Starting from one cluster and splitting it until reaching k clusters is much more cost effective than starting from hundreds of thousands of clusters and trying to reach k clusters. However, this idea contrasts with one advantage of hierarchical clustering: no predefined parameter is needed.

Divisive algorithms need another algorithm for each splitting section, which can be a flat clustering algorithm such as k-means. One significant benefit of divisive algorithms is that they have a perspective on data distribution. This is a significant benefit for these branches of clustering algorithms because in top-bottom approaches it is more likely to find strategies for undoing wrong clustering decisions, which is not the case in HACs. Therefore, clustering with side information is more applicable in divisive algorithms. Generally speaking, it is computationally more difficult to find the best splits in divisive algorithms (2^n possibilities) than the best merge in agglomerative clustering (2-permutations of n). Thus, HAC is much more widely used. The computational complexities of divisive algorithms have a wide range, from linear (in approximate non-deterministic approaches) to quintic. If the predefined number of clusters, k, is significantly smaller than the number of data points, n

($k \ll n$), the computational complexity of some divisive algorithms ($O(n)$) is significantly smaller than the computational complexity of HAC ($O(n^3)$). A detailed discussion of divisive algorithms are available in [7, 58, 141, 176, 203]. For other well-known modified versions of hierarchical algorithms, readers can refer to Chameleon [103], *Self Organizing Map (SOM)* [108], and *RObust Clustering using linKs (ROCK)* [82].

A few major drawbacks of flat clustering algorithms do not apply to HC approaches. Some advantages of this family of algorithms are that it is deterministic, has visual representation, and is parameter-free. However, inability to undo each merging or dividing step, plus undesirable high computational complexity are two main disadvantages of these algorithms. The computationally costly nature of these algorithms makes them inconvenient for large-scale problems. However, the advantages of hierarchical clustering algorithms have convinced researchers to find solutions for these drawbacks, especially the scalability problem. In order to reduce the complexity of hierarchical clustering, parallel computing, hybrid algorithms, and new hierarchical algorithm setups have been proposed. Table 2.3 shows the most widely praised HAC with their important properties under the following headings: *Computational complexity*, proven for the algorithm; *Exact* column, showing whether the solution is approximate or exact; *Cluster Shape*, showing whether the applicability of the algorithm for various clustering shapes; *Input parameter*, showing how many input parameter the algorithm needs; *Handling high-dimensional data*, showing the applicability of the solution for high-dimensional data; and *Full criteria coverage*, showing whether the algorithm can accept various merging criteria.

As can be seen in Table 2.3, only conventional approaches have the exact solution, which work with arbitrary shapes, and without input parameters, and cover all merging criteria, but they are the most-expensive algorithms. The development of new scalable variants of HAC is still an active area of research. This research pursues approaches that make HAC applicable to massive data without falling into the traps above. Next, an overview of big data clustering is presented, and is related to the current research.

2.5 Binary Codes

The data type of observed compact codes can influence the clustering approach. This research is concerned with a particular setting in which the input instances are represented with binary strings. Binary representation has several advantages such as storage efficiency, simplicity, no numerical-data-like concept of noise, and being naturally normalized.

One of the most-compact versions of data is binary representation. Such compactness is the reason behind its popularity for large-scale data representation. Mapping the data to

Clustering Algorithm	**Computational Complexity**	**Exact**	**Cluster Shape**	**Input Parameter**	**Handling High Dimensional Data**	**Full Criteria Coverage**
Conventional	$O(n^3)$	Yes	Arbitrary	No	Yes	Yes
SLINK [170]	$O(n^2)$	Yes	Arbitrary	No	Yes	No
CLINK [156, 54]	$O(n^2)$	Yes	Arbitrary	No	Yes	No
RNN [131]	$O(n^2)$	Yes	Arbitrary	No	Yes	No
ROCK [82]	$O(kn^2)$	No	Arbitrary	1	No	NA
Chameleon [103]	$O(n^2)$	Yes	Arbitrary	3	Yes	NA
BIRCH [204]	$O(n)$	No	Convex	2	No	NA
CURE [81]	$O(n^2 \log(n))$	No	Arbitrary	2	Yes	NA

Table 2.3: Major Properties of Various Agglomerative Clustering algorithms

$\{0,1\}^d$ can drastically decrease memory and the computational costs of storing, calculating distances, and thus reducing search times. The discrete nature of binary data type brings some opportunities that can be used for speeding up the pairwise nearest neighbour process. These advantages of binary code representation led us to select compact binary codes as the input space.

A notable application of binary codes is in *binary hashing* [161], which has been the topic of significant research in recent decade. The goal of binary hashing is to encode high-dimensional items, such as images, with *compact* binary strings subject to preserving a given notion of similarity. Such codes enable extremely fast *nearest-neighbour searches*, as the distance between two codes (often the *Hamming distance*) can be computed quickly using bit-wise operations implemented at the hardware level. As binary hashing is a patch to the proposed approach, the idea is briefly described in Appendix B. In this thesis, the proposed family of HAC algorithms, *Discretized Agglomerative Clustering* (DAC), is designed to work with binary data. By leveraging the discretized and bounded nature of binary representation, the proposed algorithms can achieve significant speedup, both in theory and practice, over existing solutions.

2.6 HAC in Hamming Space

Definition 4. *Given a set of records $\mathcal{X}$ with n items, where each item in $\mathcal{X}$ is a vector with the length of D, a function dis, which is responsible for finding the distance between items,* ***the clustering*** *is to find a function, f, which gives a partition of $\mathcal{X}$.*

The definition 4, that is the $\mathbb{R}^D$ clustering definition, is a general form of clustering including binary space clustering (Definition 2).

If the clustering space is $\mathbb{R}^D$, the complexity of the computing distances between all instances cannot be less than $(O(n^2D))$, where n is the number of instances, and D is the length of the vector $\mathbf{x}$ (or simply dimensions). The reason for D in computational complexity is that no matter which distance measure is used, the distance between each dimension of instances should be computed. Similar reasoning is also applicable to many $\{0,1\}^d$ distance/similarity measures, as a comparison between each dimension of instances should be applied. However, it is interesting that some of the binary distances can be computed without one-by-one dimension comparison. The most popular distance metric in $\{0,1\}^d$ with this property is Hamming distance. Although Hamming distance computational complexity is also linear to the size of the binary string (i.e., binary code or $\{0,1\}^d$ dimensions), some techniques can make it faster. One the fastest computational techniques for computing the Hamming distance is simple evaluation using a lookup table. In this technique, the evaluation is done based on a pre-computed lookup table, which is kept in memory. Other approaches such as split lookup tables and also HAMFAST [150] (i.e., based on a bit count algorithm) have also been proposed to cope with longer binary strings. Applying such techniques can eliminate the linear dependency of computational complexity on the dimensions of binary string d.

Therefore, there is an approach for mapping a dataset from $\mathbb{R}^D$ to $\{0,1\}^d$, and one can take advantage of this decrement by using fast Hamming distance computation. However, the computational costs and the quality of mapping should be considered. The mapping time should not exceed the difference between $\mathbb{R}^D$ and $\{0,1\}^d$ distance evaluations times. The quality of mapping depends on preserving the similarity. In this research, the complexity of the distance finding step is decreased to $O(n^2)$ by using Hamming-distance-evaluation approaches on the output of SimHash LSH (Appendix B). Using SimHash LSH, preserves the similarity. There are several approaches available for binary hashing to not only compress the data but also to provide a binary version of instances where fast Hamming distance computation becomes applicable. The result is used as a patch to the proposed approach, will be discussed next.

This study is mainly concerned with an efficient HAC for Hamming space. Given a set of d-dimensional binary vectors $\mathcal{B} = \{\mathbf{b}_i \in \{0,1\}^d\}_{i=1}^n$, which are compared in terms of their Hamming distances, the goal is to devise an efficient HAC algorithm that embraces all mentioned linkage criteria for binary data. Hamming distance is defined merely as the number of positions at which the corresponding bits are different. Computing Hamming distance is an extremely fast operation in modern CPUs. It is often implemented at the hardware level and can be computed using a *population count* (popcnt) operator.

One of the computational bottlenecks of HAC is finding the nearest pair of objects in each iteration. This is often done by using one of the usual implementations of priority queues such as a heap. However, by leveraging the discrete nature of binary codes, the nearest pair-search can be performed in a time that is not dependent on the number of pairs. The main idea stems from the fact that for two d binary codes, there is only a limited number of possible Hamming distances, from 0 to d. Therefore, one can partition all pairwise distances between codes into $d+1$ lists, each representing one of the possible Hamming distances. Then, starting from list 0, the closest pair can be easily retrieved by accessing the first non-empty list with the smallest index. The main problem with this approach is the updating procedure required after merging two clusters. The primary focus in the rest of this section is on proposing an efficient updating procedure. The proposed approach is called *discretized agglomerative clustering* as it is mainly designed for discrete spaces such as Hamming space.

2.7 Big Data Clustering

Beyond the sheer amount of data, several non-deterministic clustering algorithms are NP-hard problems, and many deterministic clustering algorithms are computationally expensive. K-means, one of the oldest and most well-known clustering, is an NP-hard problem even for small k's. Also, HAC, as a well-known family of algorithms, suffers from high computational complexity. There are approximate approaches for many well-known algorithms, but scaling them up without losing clustering quality still remains a problem. K-means, as an example, does not have an exact solution and is a non-convex optimization problem that is solved by alternating optimization. There is no global minimum, and cluster means are initially assigned. Then these means are used for finding better solutions, and this process continues until some convergence criterion is satisfied. As can be seen, there should be iterations for finding the local minimum. Even for this non-deterministic widely-used approximate clustering algorithm, the solution has high computational complexity.

Therefore, several approaches have been proposed for scaling up conventional clustering algorithms without much sacrifice of results' quality. As scaling-up approaches depend on the clustering algorithm, few common techniques are applicable for all clustering algorithms. On the other hand, researchers reasonably seek to invest in clustering algorithms that give better results on small datasets. Next, some approaches are briefly described that are widely used for speeding up clustering algorithms to make them applicable for big data.

One intuitive way of speeding up clustering algorithms is to distribute the computation among several machines rather than using a single machine resource. One can scale up the clustering algorithms by distributing the computation on multiple parallel machines. First, the data is *partitioned* and then *distributed* over multiple machines, then every machine performs *clustering* on its *local* data, and clustering results are *transferred* to a single machine that *combines* the results to get a *global clustering result.* The global clustering result can be *transferred back* to the local machines for *improved* local clustering. This process can be done multiple times to achieve intended results.

Because the result of clustering is highly dependent on setup data, partitioning data can change the final results of global clustering algorithms, which is a possible issue for this family of algorithms. Furthermore, as this family of approaches for speeding up clustering algorithms takes advantage of parallel machines, the cost of transferring data between machines could be another issue.

Here, some well-known parallel algorithms are mentioned without digging into their details.

- *K-means with MapReduce* MapReduce [53] is a programming framework for processing large data in a parallel way. It can be utilized to parallelize and distribute clustering algorithms. Parallel k-means [205] is a version of k-means and takes advantage of MapReduce. It shows the linear computational complexity and reasonable speed up. The result of parallel k-means is the same as for conventional k-means, although k-means itself is sensitive to initial setups.

- *Parallel DBSCAN Density-Based Spatial Clustering of Applications with Noise (DBSCAN)* [66] is famous for its power on arbitrary shapes, that is based on the density of points. DBSCAN groups points are packed together, causesing intra-cluster density to be noticeably higher than inter-cluster density. Parallel DBSCAN (or *Density-Based Distributed Clustering (DBDC)*) [95] partitions data explicitly over machines, unlike parallel k-means. One can notice another difference between these two; that is, the local clustering machines in parallel k-means are remote-server disc-based, instead of following the memory-based approach in parallel DBSCAN. The experimental results of DBDC show drastically increased clustering speed, although its results quality is only of acceptance and approximate, which means that they are inferior to the results of DBSCAN.

Other parallel and distributed algorithms have been proposed for scaling up clustering algorithms for big data. Readers can get more information about these algorithms in [39, 44, 49, 51, 53, 59, 71, 99, 100, 123, 124, 148, 147] , as well as information on many other interesting improvements and applications.

These approaches mostly concentrate on partitioning and parallelization and not on increasing the efficiency of clustering algorithms themselves. Other approaches that modify the setup of the clustering algorithm are briefly described next.

K-means is an example of a branch of algorithms that do iterations until the intended quality (or convergence threshold) is achieved. Such iterations increase computational cost and make this algorithm not applicable to big data. One optimization idea for this branch of algorithms is to limit the number of iterations (passes). *One pass* algorithms pass over the data only once. Three well-known representatives of this branch of algorithms are CLARANS [66, 142] , CURE [81], and BIRCH [204].

Clustering Large Applications based on RANdomized Search (CLARANS) is an optimized version of k-medoids clustering, which was introduced as *Partitioning Around Medoids (PAM)* in [104]. Medoids are similar to centroids or means, but they must be one of the central data points, which means that a medoid in a cluster is a data point whose average distance to other data points in that cluster is minimal. In PAM, at each iteration, one should find new medoids in each cluster, which is a very computationally costly task. The best alternative solution for PAM is CLARANS, which investigates a random sample of neighbours of the current point at each iteration. Experiments show that this strategy drastically increases the efficiency of the k-medoids algorithm.

The *Clustering Using REpresentatives (CURE)* algorithm is similar to HAC. CURE can be categorized as *representative* algorithms in which one or more points represent each cluster. Similar to other agglomerative clusterings, it treats all data points as a cluster and recursively merges them. Conventional HAC algorithms end the process when there is only one cluster, but in CURE, the clustering stops when the algorithm reaches the predefined cluster numbers. The main difference between CURE and other *representative* algorithms is that, in CURE, there are multiple representatives for one cluster, which are "well-scattered data points". This algorithm uses two data structures to perform its task: a kd-tree to store cluster representatives, and a heap to store inter-cluster distances. This algorithm uses the following steps to decrease computational time: running the algorithm on *sampled data*, *partitioning* the data, applying the algorithm *on partitions*, *merging* the results of local clusterings into global clustering, and *assigning* non-sampled data to resulting clusters of sample data according to their distance to cluster representatives.

The experimental results show that this algorithm performs clustering tasks faster than BIRCH algorithm, with a computational complexity for n data points of $O(n^2 \times \log(n))$.

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) is another approach that speeds up the clustering process. This algorithm uses a data structure called the *Clustering Feature (CF)*, and a *clustering feature tree*. CF is a vector thst includes statistical data about each cluster. It includes the zero-order, the first-order, and the third-order moments of a cluster. By using the clustering feature of each cluster, the CF-tree (a height-balanced tree) is built to hold the clustering structure. There is a threshold parameter in BIRCH, which is the diameter upper-bound of each node. Thus, if this upper-bound is set to zero, each leaf node would be a data point, which results in a very large CF-tree. BIRCH algorithm, like CURE, has a global-clustering step that clusters all sub-clusters in the leaf nodes. The main steps and optional steps of the BIRCH algorithm are as follows 1) Scanning data points and building a CF-tree (initially with a threshold of zero); 2) Condensing the tree by building smaller CF-trees and inserting their leaf entries (Optional Step); 3) Global clustering by clustering all sub-clusters; and 4) Clustering refinement by reassigning data points to clusters according to the results of global clustering.

The experiments showed that BIRCH algorithm has a lower execution time than CLARANS and a higher one than CURE. Moreover, similar to CURE, this algorithm is also robust to noise and outliers.

All approaches mentioned above modify the conventional algorithms to increase the speed of the algorithms, scale them up, and make them applicable to big data. It is worth mentioning here that the other common characteristic of these approaches is that they are non-exact algorithms, which means there is a trade-off between accuracy and speed.

One can categorize embedding and projection algorithms as approaches for scaling up clustering algorithms, but researchers believe that these approaches belong to dimensionality reduction and data-compression techniques, which are represented in their related sections.

2.8 Summary

HAC algorithms have some advantages compared to flat clustering algorithms. The former build a dendrogram that helps in visualizing the data points and clusters. No predefined parameter is needed for this family of algorithms, which is not the case in some famous flat algorithms such as k-means. Also, HAC algorithms are deterministic, in contrast to non-deterministic algorithms such as k-means, but at the cost of higher computational complexity.

In addition to their higher computational complexity, HAC algorithms also lack an optimizing strategy at each level, which is another weakness of this family. The result is a branch of rigid algorithms in which wrong clustering assignment cannot be undone, although some solutions for this drawback have been proposed. The main disadvantage of HAC algorithms is their high computational cost. Therefore, some approaches have been proposed for scaling them up by either parallelizing and distributing computation or modifying their setup. However, the latter may reduce their accuracy or/and make them parameter dependent. These algorithms have been used in several fields due to their advantages, and so increasing their performance is an active area of research.

Although HAC has been applied on categorical data, less attention has been paid to modifying their setup as the root cause of the problem and adapting them to categorical data, as this research does. This thesis addresses the aforementioned research gap specifically by exploiting the discrete nature of binary data. The goal is to decrease the required number of pairwise nearest-neighbour searches in Hamming space, which is the computational bottleneck in HAC algorithms, without introducing errors into the results.

Chapter 3

Proposed Approach

This chapter discusses the proposed approach (i.e., *Discretized Agglomerative Clustering (DAC)*) by expanding the idea of HAC algorithms and defining the roadmap from HAC to DAC-Bidirectional Arrays of Linked Lists (BALL).

As noted in Chapter 2, attempts so far to deal with the research gap in fast HAC algorithms have resulted in approaches that are either imprecise (non-exact) or limited. This proposal deals with speeding up the clustering of binary data. There is a research trend toward speeding up the clustering algorithms for complex and large-scale data (i.e., enormous numbers of instances and high dimensional data). Achieving lower bounds for any clustering algorithm is itself a moral motivation. Also, binary codes, a form of categorical data, have many advantages that can be utilized. Defining an algorithm that can decrease the computational costs of clustering binary data has two advantages: it would allow the clustering of binary datasets (unmapped) and would permit data to be transformed to a new $\{0,1\}^d$ where the memory and computational costs are lower, hopefully without losing much accuracy.

3.1 Hierarchical Agglomerative Clustering Algorithm

This chapter starts off by describing the original HAC [132], and then shows how the algorithm is improved to obtain a faster technique for binary data. Given a set of points $\mathcal{P} = \{\mathbf{p}_1 ... \mathbf{p}_n\}$ from a metric space, and a measure similarity $dis(.,.)$, Algorithm 1 shows the detailed procedure for performing HAC. First, the similarity matrix C is computed and then $n-1$ merging steps are executed, each of which combines the two most-similar clusters (or singletons).

Array A lists the merged clusters. At each merging step, the most-similar clusters are merged. Then, rows and columns related to these recently merged clusters are updated in lines 13 and 14 by using the $Sim(.,.)$ function that computes the inter-distance between two clusters. The first index, i, is considered as the index of the newly merged cluster. Also, cluster m becomes obsolete and will not be considered in the next steps. Array I also indicates the obsolete clusters. All members of the HAC family follow the paradigm of Algorithm 1, but with different inter-cluster measures of similarity. Algorithm 1 is summarized as four main steps:

1. *Distance Computation Step (DCS)*: Lines 1 to 7 of Algorithm 1. At this step, the algorithm computes the inter-instance distances and stores them.
2. *Minimum Finding Step (MFS)* or Pairwise Nearest Neighbor Search (line 10 of Algorithm 1): Obviously, the algorithm merges two clusters that are the closest pairs. MFS is responsible for finding these nearest neighbors which, is a common step in all HAC algorithms.
3. *Distance Updating Step (DUS)*: Lines 13 and 14 of Algorithm 1. After two clusters are merged, the algorithm needs to update the distance between this newly merged cluster to all other clusters. Computing updated distances is performed at this step.
4. Iterate through steps two and three until only one cluster remains.

This summarization is helpful as these common steps are going to be frequently referred to in the following.

The computational complexity of DCS is $O(n^2)$, which is the inevitable part of exact HACs as any HAC algorithm has to compute the entries of the distance matrix. That being said, DCS is not the most expensive part of HAC. The total complexity of Algorithm 1 is $O(n^3)$, because of repeatedly scanning the similarity matrix in each iteration, searching for the pairwise nearest neighbours in MFS and DUS.

In Algorithm 1, function $Sim(.,.)$ is responsible for capturing the *linkage-criteria*. The most-popular measure linkage-criteria used in practice and the literature are (Chapter 2):

- *single-linkage*: the distance between two clusters is the distance between the nearest points of these clusters:

$$Sim_{sin}(I, J) = \min\{dis(\mathbf{x}_i, \mathbf{x}_j) | \forall \mathbf{x}_i \in I, \forall \mathbf{x}_j \in J\} \tag{3.1}$$

Algorithm 1 HAC

```
1: procedure HAC($\mathbf{p}_1 ... \mathbf{p}_n$)
2:     for $i \leftarrow 1$ to $n$ do
3:         for $j \leftarrow 1$ to $n$ do
4:             $C[i][j] \leftarrow dis(\mathbf{p}_i, \mathbf{p}_j)$
5:             $I[i] \leftarrow 1$
6:         end for
7:     end for
8:     $A \leftarrow []$
9:     for $k \leftarrow 1$ to $n-1$ do
10:        $\langle i, m \rangle \leftarrow argmin_{(i,m):i \neq m, I[i]=1, I[m]=1} C[i][m]$
11:        $A.Append(\langle i, m \rangle)$
12:        for $l \leftarrow 1$ to $n$ do
13:            $C[i][l] \leftarrow Sim(i \cup m, l)$
14:            $C[l][i] \leftarrow Sim(i \cup m, l)$
15:            $I[m] \leftarrow 0$
16:        end for
17:    end for
18:    return $A$
19: end procedure
```

- *complete-linkage*: the distance between two clusters is the distance between their farthest points:

$$Sim_{com}(I, J) = \max\{dis(\mathbf{x}_i, \mathbf{x}_j) | \forall \mathbf{x}_i \in I, \forall \mathbf{x}_j \in J\} \tag{3.2}$$

- *average-linkage*: the distance between two clusters is the arithmetic average of all pairwise distances between the points of two clusters:

$$Sim_{wav}(I, J) = \frac{1}{|I|.|J|} \sum_{\mathbf{x}_i \in I} \sum_{\mathbf{x}_j \in J} dis(\mathbf{x}_j, \mathbf{x}_l) \tag{3.3}$$

It is easy to see that $Sim(.,.)$ is equal to $dis(.,.)$ for singletons in all of the linkage-criteria; that is, $Sim(\mathbf{p}_i, \mathbf{p}_j) = dis(\mathbf{p}_i, \mathbf{p}_j)$.

Next, a two-dimensional example is introduced, showing the roadmap from conventional HAC to the proposed solution: DAC-BALL.

3.1.1 HAC to DAC-BALL

Three main steps of HAC have been discussed in the previous sections. These steps are the basis of all HAC. To make these steps clear and address issues within them, an example [3] of HAC is brought in here. The example is in two-dimensional space (for demonstration simplicity), and seven data points exist in this space. The distance representative in this example is a 7×7 matrix. Each row and column of the matrix corresponds to a specific data point.

$$\begin{bmatrix} 0 & - & - & - & - & - & - \\ - & 0 & - & - & - & - & - \\ - & - & 0 & - & - & - & - \\ - & - & - & 0 & - & - & - \\ - & - & - & - & 0 & - & - \\ - & - & - & - & - & 0 & - \\ - & - & - & - & - & - & 0 \end{bmatrix} \tag{3.4}$$

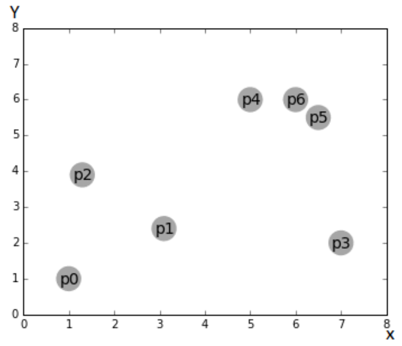


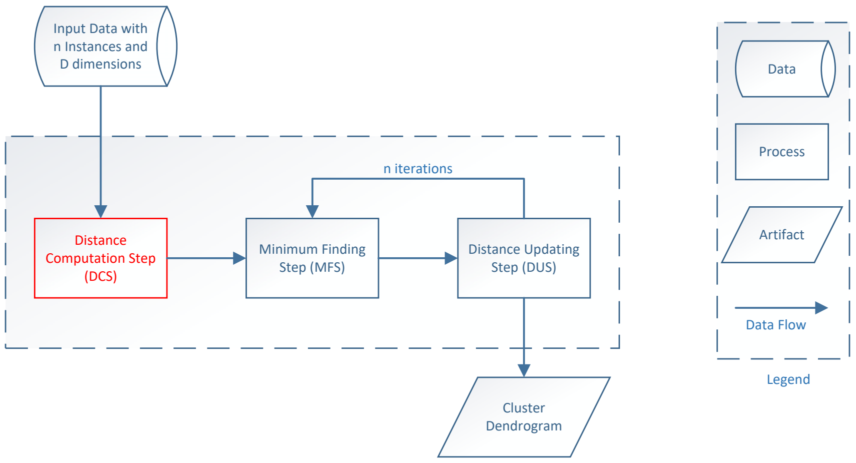
Figure 3.1: An example of 7 points in 2-D space

DCS: In DCS (Figure 3.2), the algorithm computes inner distances and fills in the matrix with computed distances in the proper row and column, so as an example, one can start from the distance computation between $p0$ and $p1$ and fill in the appropriate elements of the matrix (Figure 3.3).

Distance computation continues until all inter-instance distances become available. In this example, with the matrix as distance representation, the final distance matrix is shown in Figure 3.4.

The computational complexity of DCS is $O(n^2)$, which is the inevitable part of exact HACs as any HAC algorithm has to compute the entries of the distance matrix. That being said, DCS is not the most-expensive part of HAC. The total complexity is $O(n^3)$ because of repeatedly scanning the similarity matrix in each iteration, searching for pairwise nearest neighbours in MFS and DUS, which will be discuss in following sections:

MFS : MFS or *pairwise nearest neighbor search* is the second step of the algorithm in Figure 3.5 and another major part of HAC. Finding the closest pairs is the responsibility of this step (Figure 3.5) as these two closest clusters are going to be merged.



Input Data with n Instances and D dimensions
Distance Computation Step (DCS)
Minimum Finding Step (MFS)
n iterations
Distance Updating Step (DUS)
Cluster Dendrogram
Data
Process
Artifact
Data Flow
Legend

Figure 3.2: DCS in Agglomerative Clustering Process

$$\begin{bmatrix} 0 & 5 & - & - & - & - & - \\ 5 & 0 & - & - & - & - & - \\ - & - & 0 & - & - & - & - \\ - & - & - & 0 & - & - & - \\ - & - & - & - & 0 & - & - \\ - & - & - & - & - & 0 & - \\ - & - & - & - & - & - & 0 \end{bmatrix} \tag{3.5}$$

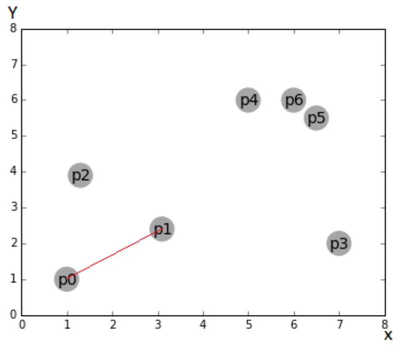


Figure 3.3: Distance Computation between $p0$ and $p1$ as an example

In the example, MFS is searching for the minimum distance within the distance matrix which is computed in the previous step of HAC (i.e., DCS). In order to find the minimum distance within a distance matrix of size $n \times n$, n^2 elements needed to be checked (49 elements in this example) (Figure 3.4).

After the closest pairs are found (Figure 3.6), they will be merged, and now the first cluster is formed. Then, the next step is to compute the distances between this newly formed cluster and all other clusters, which is the responsibility of DUS.

DUS : This distance computation (updating distances) is called DUS. DUS is the third step in HAC (Figure 3.2). After each cluster formation, distances between this newly merged cluster and all other clusters need to be updated so that the next MFS can find

$$\begin{bmatrix} 0 & 5 & 6 & 17 & 11 & 13 & 15 \\ 5 & 0 & 4 & 12 & 8 & 11 & 11 \\ 6 & 4 & 0 & 16 & 9 & 14 & 13 \\ 17 & 12 & 16 & 0 & 9 & 8 & 7 \\ 11 & 8 & 9 & 9 & 0 & 3 & 2 \\ 13 & 11 & 14 & 8 & 3 & 0 & 1 \\ 15 & 11 & 13 & 7 & 2 & 1 & 0 \end{bmatrix} \tag{3.6}$$

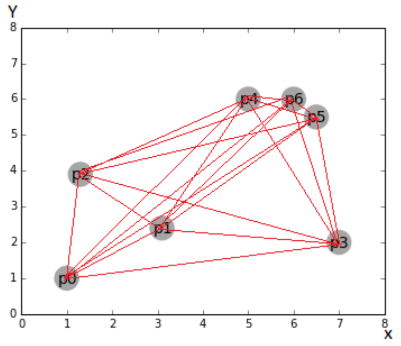
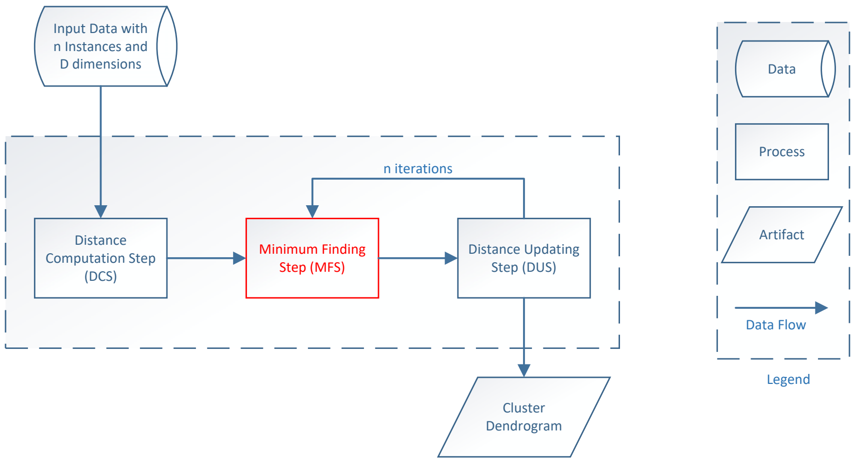


Figure 3.4: Distance Computation Finished



Input Data with n Instances and D dimensions
Distance Computation Step (DCS)
Minimum Finding Step (MFS)
Distance Updating Step (DUS)
n iterations
Cluster Dendrogram
Data
Process
Artifact
Data Flow
Legend

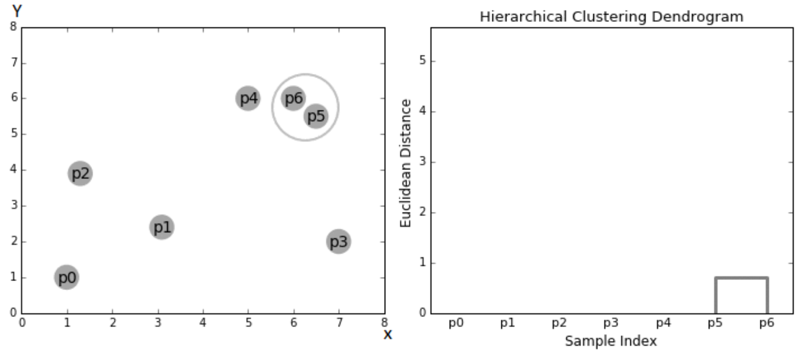
Figure 3.5: MFS in Agglomerative Clustering Process

the updated minimum distance again.

In this example, $p5$ and $p6$ are found to be the closest pairs. At DUS, the distances between the newly formed cluster, $c1$ (i.e., $p5 \cup p6$), to all other instances ($p0$ to $p4$, see Figure 3.6) are computed. Red coloured values are the ones that correspond to $p5$ and $p6$ and need to be updated. The method that is selected for DUS defines the branches of agglomerative clustering, such as single-linkage (Equation 3.1), complete-linkage (Equation 3.2), and average-linkage (Equation 3.3). After DUS, the control goes back to MFS in order to find the next-closest pairs. The agglomerative algorithm performs MFS and DUS for n iterations such that at each iteration two clusters are merged and form a new cluster until there is only one cluster in the space. The result is a dendrogram in which at each

$$\begin{bmatrix} 0 & 5 & 6 & 17 & 11 & 13 & 15 \\ - & 0 & 4 & 12 & 8 & 11 & 11 \\ - & - & 0 & 16 & 9 & 14 & 13 \\ - & - & - & 0 & 9 & 8 & 7 \\ - & - & - & - & 0 & 3 & 2 \\ - & - & - & - & - & 0 & \color{red}{1} \\ - & - & - & - & - & - & 0 \end{bmatrix}$$

(a) Distance Representation



(b) The Closest Pairs

Figure 3.6: MFS or Pairwise Nearest Neighbor Search Within Distance Matrix

level two clusters merged (see Figure 3.9).

In general, for HAC with n instances in D dimensional space, the computational complexity of distance computation (i.e., DCS) is $O(n^2D)$. In order to find the closest pairs within the distance matrix in this example, all elements of the matrix (21 elements) should be checked at each iteration. Therefore, the computational complexity of MFS is $O(n^2)$. Also, at each iteration, all distances should be updated. The total computational complexity of DUS is $O(n^2)$. If $n >> D$, the bottleneck of HAC will be the pairwise nearest neighbour search. That is the costliest part of this family of algorithms and injects the computational complexity of $O(n^3)$ into the algorithm. A two-dimensional space is used in this example for the sake of simplicity. However, most real-world problems are in high-dimensional spaces, as discussed in Chapter 2. Therefore, in such cases, the dimensionality of the data also becomes a significant issue that can not be neglected. Therefore, the computational complexity of the problem becomes $O(n^3 + n^2D)$.

One can come up with some ideas to address the computational complexity issue in

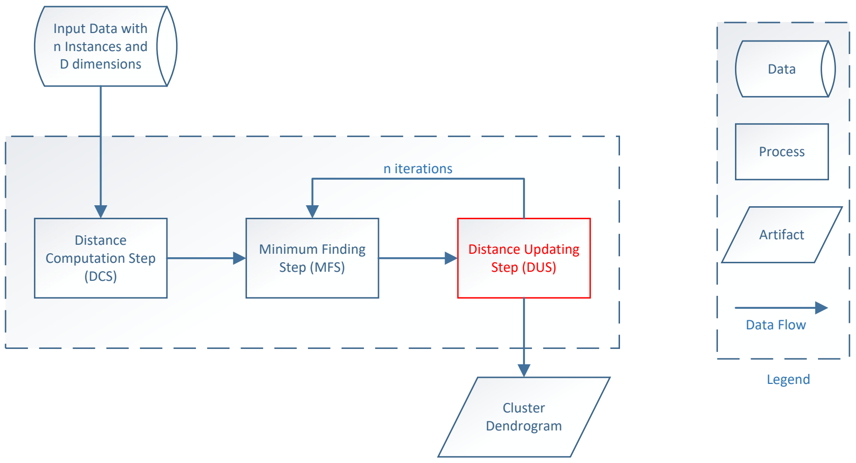

Input Data with n Instances and D dimensions
n iterations
Distance Computation Step (DCS)
Minimum Finding Step (MFS)
Distance Updating Step (DUS)
Cluster Dendrogram
Data
Process
Artifact
Data Flow
Legend

Figure 3.7: DUS in Agglomerative Clustering Process

HAC. However, as discussed in Chapter 2, most of the solutions are still costly, or not exact, or parallelized, or do not cover the full linkage criteria.

$$\begin{bmatrix} 0 & 5 & 6 & 17 & 11 & 13 & 15 \\ - & 0 & 4 & 12 & 8 & 11 & 11 \\ - & - & 0 & 16 & 9 & 14 & 13 \\ - & - & - & 0 & 9 & 8 & 7 \\ - & - & - & - & 0 & 3 & 2 \\ - & - & - & - & - & 0 & 1 \\ - & - & - & - & - & - & 0 \end{bmatrix} \tag{3.7}$$

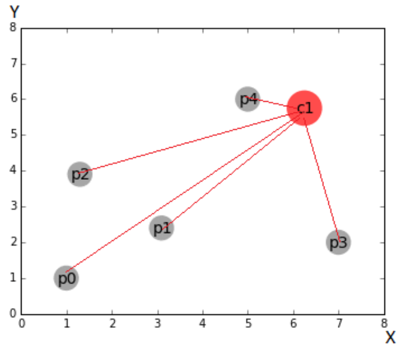


Figure 3.8: Distance Updating for the Newly Merged Cluster $c1 = p5 \cup p6$

Now, assume that the problem is transferred from a D dimensional real space to a d dimensional Hamming space by a binary hashing algorithm. The input space for HAC will be a d dimensional Hamming space. One benefit of working in Hamming space is the possibility of performing distance computations by bit-wise operations at the hardware level. This benefit can drastically increase the speed of distance computation in high dimensional spaces and promote the computational complexity of $O(n^2)$ rather than $O(n^2D)$.

At a d dimensional Hamming space, the length of binary codes is d. Therefore, the *maximum Hamming distance* between two points in this space will be d. Also, as the

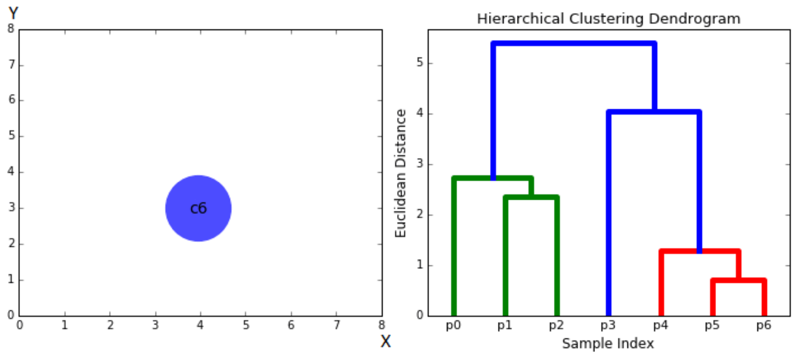


Figure 3.9: The Resulted Dendrogram

distance at each dimension of Hamming space is either 0 or 1, the distances in this space are discrete. All being said, for a d dimensional Hamming space, the distances are *discrete* and *bounded*:

$$\{0\} \cup \{1\} \cup \{2\}...\{d\} \tag{3.8}$$

Applying HAC on a discretized bounded space is the foundation of the approach proposed in this thesis, as this approach is going to take advantage of these two characteristics to address the major issues of HAC that in previous sections. Therefore, this approach is called *Discretized Agglomerative Clustering (DAC)*.

By having discrete distances bounded between 0 and d, there is no constraint to store distances in a matrix. It is possible to store them based on distance magnitude and not according to the instances indices. The magnitudes are discrete so one can store distances in $d+1$ containers such as: $d+1$ sets, $d+1$ arrays, and $d+1$ linked lists. By having $d+1$ disjoint containers, the effort to find the minimum distances (the pairwise nearest neighbours) will be eliminated, which can make agglomerative clustering much faster. However, it is also essential to know where each distance lies, as one needs to update distances at the distance-updating step of HAC. So, in addition to $d+1$ containers for storing distances, other containers are needed to store the address of each distance in memory.

The doubly linked list is a proper container for storing discrete-bounded distances. This linear data structure has the advantage of adding, removing, and updating each of its nodes in constant time. Thus, an array (with a length of $d+1$) of linked lists can be used for storing discrete distances bounded between 0 and d. In this case, all distances equal to 0 will be stored in the linked list connected to the 0th element of the array.

Assume that the example is in discrete bounded space, and the distances are bounded between 0 and 17 (see Equation 3.6 in Figure 3.4). Therefore, the array has 18 elements indexed from 0 to 17. Each element of the array is the starting point of a linked list. For instance, the linked list that is connected to the element with an index of 1 is a container for storing distances equal to 1. As can be seen in Figure 3.10, the distance between $p5$ and $p6$, which equals 1, is stored in the corresponding node (in the linked list) connected to the 1st element of the array.

This container (called the *Array of Distances' Linked Lists (ADLL)*) is constructed for storing distances during DCS. However, another container needed for DUS is also constructed to store the addresses of ADLL nodes. Figure 3.11 shows an example of the connections between a node in ADLL and two corresponding nodes in *Array of Addresses' Linked Lists (AALL).* A node in ADLL knows its related addresses in AALL, and a node in AALL knows the address of its corresponding node in ADLL. The connection between ADLL and AALL is in both directions, so this combination of containers is called Bidirectional Arrays of Linked Lists (BALL). The discretized bounded nature of data provides the possibility of proposing a new agglomerative algorithm (i.e., DAC). BALL is the data structure used with DAC. This combination of data structure and algorithm, called DAC-BALL is discussed in the next section.

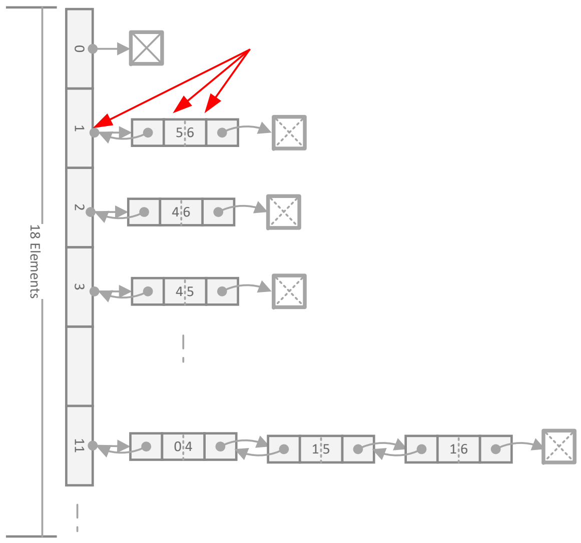
By using ADLL one can find the closest pairs without traversing the whole distance container. MFS can be performed by checking only the first element of the ADLL (see Figure 3.10). This change can drastically decrease the computational complexity of MFS from $O(n^2)$ to $O(d)$. DUS can be performed by removing all nodes in ADLL and AALL related to the closest pairs and adding new nodes to ADLL and AALL after computing new distances. The computational complexity of this step by DAC-BALL becomes $O(n)$, which is similar to the complexity of DUS in HAC.

3.2 Formalizing DAC-BALL

In this section, the idea of using the list of distances discussed above is formalized and the required data structures and algorithms is described. An array of linked lists for storing all pairwise distances is utilized in the proposed approach. The length of the array is $d+1$, which the number of possible Hamming distances. Each entry of the array, such as i, points to a linked list that stores a pair of indices, where each pair refers to two clusters with distance i from each other. As mentioned, this array of linked lists is called the *Array of Distances' Linked Lists (ADLL).* Each entry of linked lists (called a *node*) includes indices of the first and the second clusters (i, j) related to their calculated distance $(dis(i, j))$.

$$\begin{bmatrix} 0 & 5 & 6 & 17 & 11 & 13 & 15 \\ - & 0 & 4 & 12 & 8 & 11 & 11 \\ - & - & 0 & 16 & 9 & 14 & 13 \\ - & - & - & 0 & 9 & 8 & 7 \\ - & - & - & - & 0 & 3 & 2 \\ - & - & - & - & - & 0 & 1 \\ - & - & - & - & - & - & 0 \end{bmatrix}$$

(a) Distance Representation



(b) The Closest Pairs

Figure 3.10: The Distance between $p5$ and $p6$ Shown in Matrix and Arrays of Linked Lists

Now suppose the clustering algorithm is about to merge two points (or clusters), such as i and j, which have the smallest pairwise distance. The problem is that after merging, indices i and j become obsolete as these points do not exist anymore. One naive solution would entail iterating through all nodes in ADLL and removing those that contain either i or j as one of their entries, but this requires $O(n^2)$ time per merging, which is costly. The idea proposed here is to have another array of length n, called *Array of Addresses' Linked Lists (AALL)*, where each entry, such as AALL[i], is again a linked list that saves the addresses of all nodes in ADLL that have i as one of their indices. By using this new array, when merging points i and j, the algorithm can simply iterate through the linked list stored in AALL[i] and AALL[j] and find the nodes in ADLL that must be deleted. By doing this, the time for deleting obsolete nodes is reduced from $O(n^2)$ to $O(n)$ because

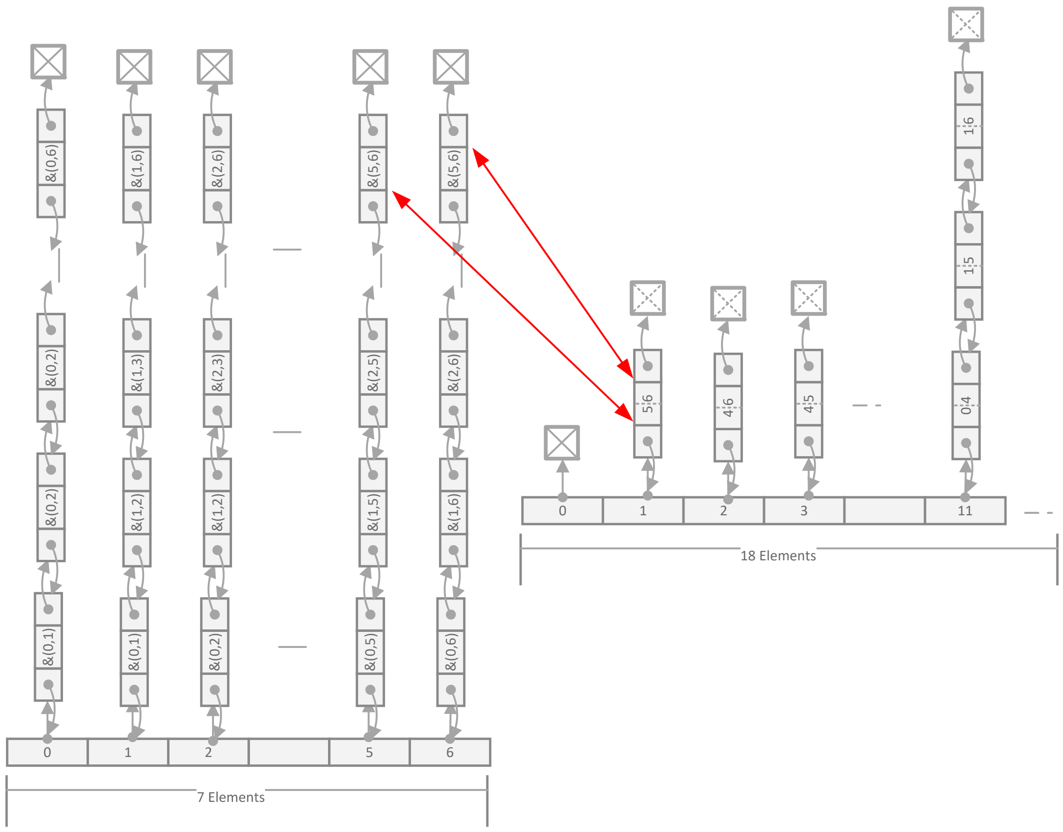


Figure 3.11: The Bidirectional Connection Between ADLL and AALL

the length of AALL[i] and AALL[j] is at most n. However, this gain comes at the cost of storing an extra array.

The ath linked list of ADLL includes all pairs of data points whose distances are a. Therefore, the lengths of the array depend on the maximum distance between two data points, which for binary vectors, is the length of the codes, d. The number of nodes in ADLL for each index of the array depends on the number of possible distances. For example, for binary data, the number of possible distances between two data points is limited to the length of binary codes. The length of the array in AALL is the number of data points, n. The maximum length of a linked list in AALL is n. The ith linked list of AALL includes all addresses of nodes in ADLL related to i. Thus, the address of every node in ADLL that stores pairwise distances between i and another cluster m is stored in the ith and m linked lists of AALL.

To illustrate, assume the distance between i and j is dis_{ij}, so, in the dis_{ij}th linked list of ADLL, there is a node, say ND_{ij}, that stores i and j. One node (NA_i) in the ith linked list of AALL has a pointer to ND_{ij}. Similarly, one node (NA_j) in the jth linked of AALL list has a pointer to ND_{ij}. Also, ND_{ij} has a pointer to each NA_i and NA_j.

Here, each ADLL node includes indices of the first and second clusters (i, j) related to the pairwise distance $(dis(i, j))$, and the addresses of corresponding nodes in AALL. The connection between the nodes in AALL and ADLL are shown in Figure 3.12. Although both doubly or singly linked lists can be utilized, to take full advantage of linked lists and to have the fastest possible way to add and remove nodes, doubly linked lists are used.

While proposing some similar and less-complicated data structures is possible if the focus is on one of the linkage-criteria, the goal here is to propose a general approach that captures all distances. However, such data structures can make the algorithm linkage-specific. The proposed approach can be seen in Algorithm 2.

Let us first define the functions *AmendNode* and *ClearNode* used in Algorithm 2. Each node in ADLL includes four entities: two indices which refer to two clusters, say i and j, and two addresses (see Figure 3.12). Algorithm 2 starts by creating the node (ND_{ij}) in ADLL (line 4). Using the address of this recently created node, two nodes are appended to AALL: one in the ith linked list of AALL(NA_i), and another in the jth linked list of AALL (NA_j), both having a pointer to ND_{ij}. Finally, the function *AmendNode* amends the address entities in ND_{ij} by having the address of NA_i and NA_j.

The distance between i and j is dis_{ij}. So in the dis_{ij}th linked list of ADLL, there is a node, say ND_{ij}, that stores i and j. One node (NA_i) in the ith linked list of AALL has a pointer to ND_{ij}. Similarly, one node (NA_j) in the jth linked of AALL list has a pointer to ND_{ij}. When one decides to remove ND_{ij}, NA_i and NA_j should also be removed. *ClearNode* takes two cluster indices (i and j), and removes ND_{ij}, NA_i, NA_j.

In the following, all four simple steps in BALL are described:

1. DCS: Lines 1 to 10 of Algorithm 2. Here, similar to conventional HAC, the distances between all pair of instances are computed. After each pairwise distance computation, one node to ADLL and two corresponding nodes to AALL are added. Now the address entities in the ADLL node are amended.

2. MFS: Line 13 of Algorithm 2. Here in BALL, one does not search all pairwise distances to find the closest pairs. The first node in ADLL is the pairwise nearest neighbour.

Algorithm 2 DAC-BALL

```
 1: procedure DAC($\mathbf{b}_1 ... \mathbf{b}_n$)
 2:     for $i \leftarrow 1$ to $n$ do
 3:         for $j \leftarrow 1$ to $n$ do
 4:             $ADLL[Sim(\mathbf{b}_i, \mathbf{b}_j)].AddNode(i, j)$
 5:             $AALL[i].AddNode(Address[ADLL[Sim(\mathbf{b}_i, \mathbf{b}_j)]])$
 6:             $AALL[j].AddNode(Address[ADLL[Sim(\mathbf{b}_i, \mathbf{b}_j)]])$
 7:             $ADLL[Sim(\mathbf{b}_i, \mathbf{b}_j)].AmendNode(Address[AALL[i]])$
 8:             $ADLL[Sim(\mathbf{b}_i, \mathbf{b}_j)].AmendNode(Address[AALL[j]])$
 9:         end for
10:     end for
11:     $A \leftarrow []$
12:     for $k \leftarrow 1$ to $n - 1$ do
13:         $\langle i, m \rangle \leftarrow argmax_{(i,m)} ADLL$
14:         $A.Append(\langle i, m \rangle)$
15:         for $l \leftarrow 1$ to $n$ do
16:             $T[i][l] \leftarrow Sim(i \cup m, l)$
17:             $AALL[i].ClearNodes(i, l)$
18:             $AALL[m].ClearNodes(m, l)$
19:             $ADLL[T[i][l]].AddNode(i, l)$
20:             $AALL[i].AddNode(Address[T[i][l]])$
21:             $AALL[l].AddNode(Address[T[i][l]])$
22:             $ADLL[T[i][l]].AmendNode(Address[AALL[i]])$
23:             $ADLL[T[i][l]].AmendNode(Address[AALL[l]])$
24:         end for
25:     end for
26:     return $A$
27: end procedure
```

3. DUS: Lines 16 to 23 of Algorithm 2. The distance computation between a newly merged cluster and all other clusters is similar to that in conventional HAC, but the distance-updating step in the proposed approach is more complicated than that in conventional HAC. As distances are stored based on their extent, it is essential to remove all obsolete distances from ADLL. Assume that cluster $i \cup m$ is the newly merged cluster, and the distances between this cluster and cluster l should be updated. In some HAC methods, namely single-linkage and complete-linkage, the updated distance ($dis_{(i \cup m)l}$) is either dis_{il} or dis_{ml}. Therefore, some nodes can be reused. However, in most HAC methods, this is not the case. So, as it can be seen in line 17 and 18 of Algorithm 2, after the distance computation between $i \cup m$ and l, all nodes related to dis_{il} and dis_{ml} are removed from AALL and ADLL. Then one node is added to ADLL for $dis_{(i \cup m)l}$ and two corresponding nodes are added to AALL.

4. Steps two and three are iterated through until only one cluster remains.

By comparing algorithms 1 and 2 one can notice that the distance computation (i.e., line 4 of both algorithms) is similar except for the fact that there is no matrix representation of distance in Algorithm 2. Instead, distances are represented as an array of linked lists. Finding the closest pair in BALL is performed without searching, as the first node in ADLL holds the nearest-neighbour pairs (line 11 of the Algorithm 2), helping eliminate the need to traverse over all elements of the distance matrix to find the minimum distance, which is the convention in HAC. Just as importantly, bidirectional addressing between ADLL and AALL allows us to apply *trimming*, which is the process of removing obsolete information from a distance array (or any data structure used for storing distances). Trimming enables progressively reducing the required storage as the algorithm goes higher in the dendrogram. Trimming is applied to BALL by removing all nodes related to one of the nodes corresponding to the closest pair at each step, as can be seen in line 18 of Algorithm 2. In the conventional HAC, because the MFS is performed on the distance matrix, trimming speeds up the whole clustering process. However, in BALL, trimming reduces memory usage at each clustering iteration only.

It is possible to use unidirectional addressing, in which there would be no pointer from ADLL to AALL. However, by applying bidirectional addressing instead of unidirectional addressing, the worst case of accessing all related nodes for each instance will be $O(n)$. Also, each instance can be removed (i.e., trimming) by removing its distance node-representative with the computational complexity of $O(n)$.

What it is gained by choosing this approach is the removal of MFS. However, some additional computational costs are injected by using additional procedures and data structures. Removing two linked lists during the k-th iteration imposes $n - k$ computation.

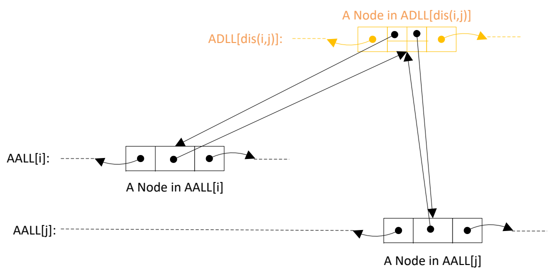


Figure 3.12: The Connection between linked lists

Adding a linked list at each iteration with updated distances costs $n-(k+1)$. The total computation for these added costs is

$$n(3n-1)-\frac{(3n(n+1))}{2} \tag{3.9}$$

Considering the distance computation ($O(n^2)$), the total computational complexity of BALL is $O(n^2)$

In summary, by removing pairwise nearest neighbour searching from HAC, the proposed discretized agglomerative clustering (DAC) achieves the computational complexity of $O(n^2)$. Therefore, if the dataset is in $\mathbb{R}^D$, by transforming data to $\{0,1\}^d$ using Hamming distance and applying BALL, the order of complexity is decreased from $O(n^2D)$ (i.e., distance computation) $+O(n^3)$ (i.e., HAC) to $O(n^2)$.

3.3 Summary

In this chapter, the conventional HAC is discussed in detail. The flaws of this family of algorithms are described using an example of applying HAC on a two-dimensional seven-point dataset. The discretized bounded space is then introduced, and its potential for improving the computational complexity of HAC is illustrated with the example. The combination of the data structure and algorithm (i.e., Discretized Agglomerative Clustering (DAC) and Bidirectional Arrays of Linked Lists (BALL)) are introduced, showing their ability to decrease the computational complexity of HAC in discretized bounded space. The proposed approach uses a different method for storing distances and amalgamates two primary steps of HAC. DAC-BALL fills a gap in the research, being an exact clustering algorithm with full coverage of linkage criteria that reaches the known lower-bound of hierarchical agglomerative clustering algorithms.

Chapter 4

Empirical Studies

In this chapter, the experimental results of the proposed clustering algorithm and its corresponding data structure are presented, along with a comparison of the efficiency and accuracy based on evaluation measures presented in Section 4.1. The efficiency and accuracy of the algorithm confirm its total performance. The efficiency is related to resources that have been used to run a clustering algorithm such as running time and memory usage. The accuracy of an algorithm is based on the quality of clustering results. In Section 4.2, the environment where the current experiments were performed is briefly explained, and in Section 4.3, the datasets involved in the experiments are described. In Section 4.4, some implementation details are presented. Section 4.5 shows the results of experiments on the datasets and related discussion.

4.1 Clustering Evaluation Measures

The quality of clustering algorithms can be measured by intra-cluster, inter-cluster, or both similarity values. As can be deduced from these names, intra-cluster similarity refers to the similarity of items within clusters, and inter-cluster similarity refers to the similarity of records between different clusters. Except for quality measures of clustering, one should evaluate the resources–the time and space (i.e., memory)–needed for performing cluster analysis.

One can categorize evaluation measures for clustering into *internal* and *external measures*. As clustering is an exploratory data analysis task, there is no a priori knowledge about the parameters of cluster analysis. Therefore, the clustering objective functions try to optimize the internal clustering evaluation measures with no information about items except their feature values. In other words, the objective functions aim to minimize the intra-cluster similarity and maximize the inter-cluster similarity values just by using input features. This type of evaluation measure–which is based just on feature values–is called internal evaluation. In Appendix C, some data similarity measures related to the current research are described.

The capability of clustering algorithms varies even when they are tested on the same datasets. The performance of clustering is divided into two scales [129]: 1- *Compactness*, which shows how close the data points are within clusters, and 2- *Separation*, which shows the separation between clusters. These scales are relevant when one talks about internal evaluation measures (there are no labels for data points).

As mentioned, cluster evaluation includes two variations: internal and external measures. Internal measures give negative points to inter-cluster similarities and positive points to intra-cluster similarities. One can find examples in which high scores for internal measures for cluster evaluation do not necessarily mean good clustering performance. In contrast, *external clustering measures* analyze the closeness of clustering to some references (i.e., the gold standard). External clustering measures come into account when clustering is done, and an external audit is needed to check the performance of the clustering approach when ground truth exists for data points.

No one can claim that one specific clustering algorithm is the absolute best clustering method and fits every clustering problem. Likewise, no one can claim that there is an external clustering validity measure that is best for the comparison of clustering algorithms and optimal for every problem. Yet, one can identify some properties which help external measure selection for clustering validity: *Metric Properties*, *Normalization* (Range), *Sample Size Dependency*, and *Coarsening and Refining Clusters Dependency.* More discussion about these properties is beyond the scope of this research but can be found in clustering literature.

External evaluation measures are categorized into three classes (Appendix C). In this research, the main focus is on the *pair-counting* class of external evaluation measures. *Rand Index (RI)*, *Adjusted Rand Index (ARI)*, and *purity* are utilized, as they are the most well-known and most-used measures for validating clustering algorithms. Here, these evaluation measures are recalled and have been defined in Appendix C.

Having a set of clusters, C, including i cluster, and a set of classes, G, including j classes, the *purity* is calculated by

$$Purity(C,G) = \frac{1}{n}\sum_{i}\max_{j}|C_i \cap G_j| \tag{4.1}$$

Purity lies between [0, 1] when 0 shows no agreements between the gold standard clustering results (i.e., G , or *ground truth*) and clustering results (i.e., C), and 1 shows complete agreement between the results and gold standard results.

$$RI = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.2}$$

Rand Index (RI) lies in the nominal range of [0, 1] when 0 shows no agreements between data labels (i.e., gold standard clustering results) and clustering results, and 1 shows complete agreement between results and class labels. *True Positive (TP)* and *True Negative (TN)* are correctly clustered elements, and *False Positive (FP)* and *False Negative (FN)* are wrongly clustered elements.

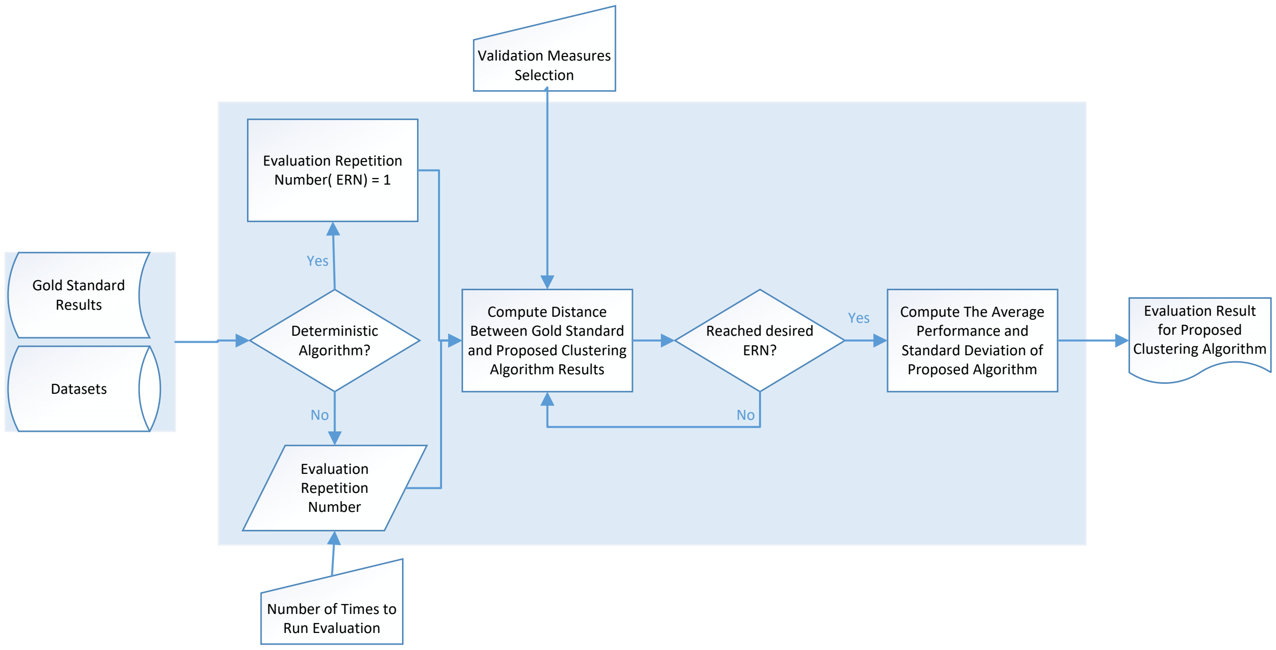
$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}} \tag{4.3}$$

The *Adjusted Rand Index (ARI)* is the corrected-for-chance version of the RI. Though the RI may only yield a value between 0 and 1, ARI can produce negative values if the index is less than the expected index. n_{ij}, a_i, and b_j are values from the contingency table (Appendix C), and n is the number of data points.

Figure 4.1 shows the process of external cluster evaluation that is performed by using the distance between gold standard results (assuming that this clustering results as a "correct" one), i.e., G and the clustering results of a proposed algorithm, i.e., C. If a proposed clustering algorithm is not deterministic (e.g., k-means results depend on the starting point of clustering), one needs to find the distance (i.e., external measure) several times and then calculate their average of them to find the average performance of the proposed algorithm. The next step is to utilize the clustering algorithm on several datasets and evaluate external measures on them. By using these distances, the mean and standard-deviation of performance for this clustering algorithm can be extracted.

4.1.1 Algorithm's Performance

The performance of a clustering algorithm is analyzed based on two criteria [10]: *scalability* and *sensitivity*. The scalability tests of an algorithm show the effect of dimensionality, the number of data-points, and the number of clusters on the clustering algorithm. The scalability is evaluated regarding the algorithm's execution time and memory usage.



Validation Measures Selection
Evaluation Repetition Number(ERN) = 1
Gold Standard Results
Datasets
Yes
Deterministic Algorithm?
No
Evaluation Repetition Number
Number of Times to Run Evaluation
Compute Distance Between Gold Standard and Proposed Clustering Algorithm Results
Reached desired ERN?
Yes
No
Compute The Average Performance and Standard Deviation of Proposed Algorithm
Evaluation Result for Proposed Clustering Algorithm

Figure 4.1: Clustering Evaluation Process

The memory usage shows the amount of memory the algorithm and data structure uses to perform clustering tasks. This memory includes storing the data if needed, the distances between data-points, and the data structures used, such as an array of linked-lists or the number of nodes in all linked-lists.

The execution time is the total clock time that is used by each algorithm to process the whole data. The process includes all tasks–from pre-processing of raw data to producing final clustering results. The data transformation (mapping data from $\mathbb{R}^D$ to $\{0,1\}^d$) is a part of this process that is also considered in the algorithms running time. The execution time is evaluated with different sizes of data, dimensions, and numbers of clusters. This evaluation is performed on multiple synthetic and well-known datasets. The execution time is analyzed on some combination of different dimensionalities and a varied number of data-points.

The sensitivity of a clustering algorithm shows the effect of the necessary parameters of the algorithm on clustering results. It analyzes the correlation between clustering input parameters and clustering quality. The result of this analysis can give the best range of input parameters for an algorithm, which depends on the nature of a dataset. One advantage of HAC is that this family of algorithms is parameter-free. Thus, in the current research, a sensitivity analysis has not been performed.

After this description of the evaluation process and metrics for the proposed approach, in the next section, the datasets used for performing the experiments are explained. However, before that, the methods and materials employed in the experiments are described.

4.2 Study Setup

In this section, the environment used in the experiments is described, including the programming language and libraries, development tools, and hardware for testing. Although datasets can be a subset of methods and materials, the dataset explanation are left for a specific section.

The DAC algorithm was developed in C++ version 11 and compiled with the GCC package that also includes some other GNU programming languages (i.e., C, Java, Fortran, Objective C, and Ada 95). Several standard libraries come with this programming language, including useful implementations such as container classes (e.g. list, queue, vector). Although these libraries can speed up the development process, they have not been used in the proposed algorithm. Running time comparison is one of the primary evaluation goals. Therefore, DAC and other algorithms are built subject to these experiments from scratch. The implemented algorithm uses neither multi-threading nor parallelism, although it is possible to do so, and is area for the future work.

One of the development tools used in the current research is the Eclipse IDE for C/C++ developers. Eclipse is an open-source full-feature Integrated Development Environment (IDE). This IDE is supported by IBM and has other programming languages (such as Java, PHP, Python, Perl) under its umbrella. Eclipse is a cross-platform IDE that can run under Windows, Linux and Mac OS. Several features and utilities such as refactoring tools, code formatting, syntax checking, EGit, task lists, source code navigation, and static code analysis help the developer during the application-development process.

Also, GDB is utilized for code debugging. GDB (the GNU project debugger) is a debugging tool which helps a developer to see what is going on inside the program while it executes and what the program is doing when it crashes. Apart from C++, several other languages such as Pascal, ADA, C, Objective C can use GDB as a debugger. It is also worth mentioning that MATLAB has been used in some stages of the research to double-check some intermediate or final clustering results.

Experiments were conducted on a single core of the CIUW2 computer server that is hosted by the CST (coding and signal transmission) group. CIUW2 has 32 physical CPU cores–Intel XEON X7550 based running at 2.00 GHz–and equipped with 264 GB RAM. The operating system on this machine is a 64-bit Scientific Linux.

4.3 Dataset Descriptions

DAC and the corresponding data structure (i.e., BALL) are evaluated by applying them to various datasets. It is essential to validate the algorithm on a variety of datasets to make sure the desired objectives such as efficiency are achieved without losing much quality of clustering. Two broad types of datasets are used in the literature: synthetic and real-world datasets.

Synthetic datasets have been used to evaluate performance objectives for the proposed clustering algorithm. These datasets are generated with various degrees of dimensionality, compactnesses, and separation. In this research, ten synthetic datasets are also generated. In addition to the performance evaluation, these datasets are generated to check the chaining and outlier effect on both $\mathbb{R}^D$ and transformed $\{0, 1\}^d$ in some versions of HAC. The specifications for these synthetic datasets are shown in Table 4.1.

Real-world datasets are gathered from the UC Irvine (UCI repository [5]), University of Toronto (CIFAR dataset [110], IRISA (TEXMEX dataset [97]), and MNIST [116] (combination of two NIST handwritten digits). The real-world datasets used in this research are IRIS, Ecoli, Image Segmentation, CIFAR-10, TEXMEX, and MNIST. Table 4.1 shows detailed specifications of these datasets.

All these datasets are used for evaluating DAC and testing its performance for various sizes (number of data-points), dimensions of data (number of features). The first step of clustering is pre-processing. All raw data must pass through this step to gain a precise and organized format so as to to become coherent input for the clustering algorithm, which guarantees the quality of results. Next, more details about this step on datasets are briefly discussed.

4.3.1 Data Pre-processing

As mentioned, data pre-processing is a major task in delivering organized and precisely formatted data as the input of the clustering algorithm (or any other data science applications). It improves the quality and consistency of clustering results. It is noted that cleaning the data, feature selection or extraction, and normalization are part of data pre-processing.

Dataset Properties					
Dataset Name	**Data Type**	**Number of Instances**	**Number of Attributes**	**Number of Classes**	**Missing Value**
Iris	Real	150	4	3	No
Ecoli	Real & Integer	327	7	5	No
Image-Segmentation	Real & Integer	2310	19	7	No
CIFAR-10	Real	50000	3072	10	No
TEXMEX-ANNSIFT10K	Real & Integer	10000	128	-	No
MNIST	Real	60000	784	10	No
Synthetic	Real	10000	128	10	No

Table 4.1: Dataset Specifications

- *Data Cleaning:* All datasets used in this experiment were cleaned, and had no missing value or outliers.

- *Data Normalization:* Normalization is applied to data to transform them into a range between a specific range (e.g., between 0 and 1). All clustering algorithms used in the experiments are based on pairwise distances. The distance between two pairs of points is based on the distances of all dimensions (features). By normalization, the possible bias of any specific dimension because of its higher magnitude is removed. Thus, normalization is necessary in order to improve the quality of results in the experiments. The min-max normalization technique was chosen for this research. A linear transformation from original data to the selected range of output was performed in this technique (e.g., setting the minimum value of a dimension to 0 and the maximum value to 1). Given a data with the range X_{min} to X_{max} for one specific feature (dimension), the new range $Xnew_{min}$ to $Xnew_{max}$, and the old value X, the range of the feature should be restricted between $Xnew_{min}$ and $Xnew_{max}$. A new value, X', for the feature is calculated by:

$$X' = Xnew_{min} + \frac{(X - X_{min})\,(Xnew_{max} - Xnew_{min})}{X_{\max} - X_{\min}} \tag{4.4}$$

  X and X' are the old and the new values for this specific dimension, respectively.

- *Feature Selection or Feature Extraction:* No feature selection or feature extraction techniques for $\mathbb{R}^D$ data were used, although for transforming data to $\{0,1\}^d$ simhash

LSH utilized, which can be considered a feature extraction or data compression technique.

The output of these three pre-processing steps was precise and organized data that can serve as the input for clustering algorithms. These pre-processing tasks for all the datasets and the clustering algorithms were performed on the pre-processed data one at a time to assess the performance and the quality of algorithms. The experimental results are discussed in the following sections.

4.4 Implementation Details

Multivariate Data Representation

Akin to many multivariate computational tasks, clustering deals with objects that have multiple features (variables or attributes), which are represented as numerical values. Clustering is applied to a representation of the multivariate numerical instances of the object. This representation of data is either naturally numerical or mapped to a numerical space. One way of representing data is storing the data in an $n \times d$ data matrix (where n is the number of data points, and d is the dimension of data). Whenever two points must be merged, the distance must be calculated instantly. If the clustering approach is dissimilarity-based, storing the distance matrix is more intuitive. This approach itself has two means of implementation: one is having an $n \times n$ matrix, often a triangular matrix, whose top level is pseudo-index-sorted distances. The bottom triangle is zero. One benefit of this triangular matrix compared to a filled-square matrix is that $\forall d(i, j) \in U : i < j$, which has benefits in clustering consistency (the upper triangular matrix is typically denoted with U). The second is the possibility of converting the triangular matrix to a row matrix (i.e., $1 \times n$), which is better than a triangular matrix memory-wise, but as always, there is a trade-off between computational costs and memory costs. When using a triangular matrix to find a point (which is the distance between two data-points or clusters), only their indices are used, considering the assumption that $i < j$, and thus easily pass over all elements of the matrix. The opposite direction is also easy if a distance is found(i.e., an element of the distance matrix). Finding two corresponding data-points or clusters is straightforward, as they are merely a row and a column of that element. However, in a row matrix, knowing the element number, some computation is needed to find the related i and j (i.e., the indices of corresponding data-points or clusters). Next shown is how an upper triangular matrix to a row matrix is mapped. Although this method is not used in DAC, it is utilized in implementing the conventional approach to agglomerative clustering.

$$DistanceMatrix = \begin{pmatrix} d_{11} & d_{12} & . & . & . & d_{1n} \\ & d_{22} & . & . & . & d_{2n} \\ & & . & & & \\ & & & . & d_{ij} & \\ & 0 & & & . & \\ & & & & & d_{nn} \end{pmatrix} \tag{4.5}$$

This upper triangular matrix is equivalent to:

$$\begin{gathered} DistanceArray = \begin{bmatrix} d_{12} & d_{13} & \cdots & d_{ij} & \cdots & d_{(n-1)n} \end{bmatrix} \equiv \\ \begin{bmatrix} d_1 & d_2 & \cdots & d_k & \cdots & d_{(n-1)n} \end{bmatrix} \end{gathered} \tag{4.6}$$

The total number of non-zero elements in a triangular distance matrix is $\Sigma i = n \times (n-1)/2$, and zero elements: $n \times (n+1)/2$. For a large n, this triangle can waste allocation space (obviously for static allocation). Ideally, only non-zero elements are stored, and only notionally, leaving out the zero (or computable) part. Only the formula is mentioned; its derivation is skipped.

$$i = \lfloor n + 1/2 - \sqrt{(n^2 - n + 1/4 - 2 \times (k-1))} \rfloor \tag{4.7}$$

$$j = k - (i-1) \times (n - i/2) + i \tag{4.8}$$

$$k = j - (3 \times i)/2 + i \times n - i^2/2 \tag{4.9}$$

For original space or conventional HAC in $\{0,1\}^d$, distance row matrix representation is used in the experiments. By passing over each element of the distance array and finding the minimum(s) and its index k, the indices of corresponding data-points or clusters i, j can be computed. All the points (clusters) corresponding to i and j are found with equations 4.7, 4.8 and 4.9.

In the proposed approach for $\{0,1\}^d$, the distances are saved rather than the raw data. However, neither type of matrices is saved. Instead, the representation of pairwise distances is an array of linked-lists. The discrete nature of $\{0,1\}^d$ introduces the opportunity of having discrete and bounded distances. As an example, for any two data points with the binary code length of d, the Hamming distance is discrete and between zero and d. Therefore, the total number of possible Hamming distances is d. This total gives the opportunity to store distances in an array (with the length of d) of linked-lists instead of distance matrices (or any corresponding geometrical representation). Each node in a linked-list is a distance between two data-points, and the distance is the index of corresponding elements of the array. For instance, all distances equal to d_{bi} are stored in a linked-list related to the d_{bi} element of the array. Having this representation, the core of the proposed approach has two benefits: 1) constant-time pairwise nearest-neighbour searching (finding the closest pair of data-points or clusters), and 2) constant-time pairwise-distance removal. These benefits will be discussed in more detail in subsequent sections.

Optimization by Trimming

One main step of HAC is the distance-updating step (DUS), performed by recalculating the distance between the newly merged cluster and all other clusters or instances. Removing some futile parts of a distance array (or any distance representative) is called *trimming*, which has been utilized in the implementation of the current research. But if all inter-instance distances are necessary for DUS equations such as 3.3, 3.1, and 3.2, there will be no futile information. Interestingly, inter-cluster (not inter-instance) distances and the number of instances in each cluster are all vital information for any HAC method.

Given a set of records in three clusters, A, B, and C, with na, nb, and nc items, respectively, where $A = \{\mathbf{a}_1...\mathbf{a}_{na}\}$, $B = \{\mathbf{b}_1...\mathbf{b}_{nb}\}$ and $C = \{\mathbf{c}_1...\mathbf{c}_{nc}\}$, a cluster D with nd items, which is the cluster formed by merging A and B, and a function dis, which is responsible for finding the distance between items, the distances between these clusters are:

$$\begin{gathered} Sim_{AC} = \min\{dis(\mathbf{a}, \mathbf{c}) | \forall \mathbf{a} \in A, \forall \mathbf{c} \in C\} \Rightarrow \\ \forall \mathbf{a} \in A, \forall \mathbf{c} \in C, \exists \mathbf{a}_{ac} \in A, \mathbf{c}_{ac} \in C | dis(\mathbf{a}_{ac}, \mathbf{c}_{ac}) \leq dis(\mathbf{a}, \mathbf{c}) \end{gathered} \tag{4.10}$$

and

$$\begin{gathered} Sim_{BC} = \min\{dis(\mathbf{b}, \mathbf{c}) | \forall \mathbf{b} \in B, \forall \mathbf{c} \in C\} \Rightarrow \\ \forall \mathbf{b} \in B, \forall \mathbf{c} \in C, \exists \mathbf{b}_{bc} \in B, \mathbf{c}_{bc} \in C | dis(\mathbf{b}_{bc}, \mathbf{c}_{bc}) \leq dis(\mathbf{b}, \mathbf{c}) \end{gathered} \tag{4.11}$$

and after merging clusters A and B:

$$\begin{gathered} D = A \cup B = \{\mathbf{a}_1...\mathbf{a}_{na}\} \cup \{\mathbf{b}_1...\mathbf{b}_{nb}\} = \{\mathbf{a}_1...\mathbf{a}_{na}, \mathbf{b}_1...\mathbf{b}_{nb}\} \\ Sim_{DC} = \min\{dis(\mathbf{d}, \mathbf{c}) | \forall \mathbf{d} \in D, \forall \mathbf{c} \in C\} \\ \forall \mathbf{d} \in D, \forall \mathbf{c} \in C, \exists \mathbf{d}_{dc} \in D, \mathbf{c}_{dc} \in C | dis(\mathbf{d}_{dc}, \mathbf{c}_{dc}) \leq dis(\mathbf{d}, \mathbf{c}) \end{gathered} \tag{4.12}$$

$$Sim_{DC} = \min(Sim_{AC}, Sim_{BC}) \tag{4.13}$$

The proof of Equation 4.13 is evident by contradiction considering that the two clusters A and B, are disjoint sets and have no elements in common, and D is the union of these two clusters. Therefore, for updating the distance of a newly merged cluster (i.e., here D) with other clusters in single-linkage HAC, one merely needs to find the minimum distances between the particular cluster(i.e., C) and both A and B. Thus, the only information needed to update the distance between any cluster C and the newly formed cluster $A \cup B$ is the Sim_{AC} and Sim_{BC}. Therefore, all inter-instance or inter-cluster distances corresponding to A and B, before merging, become obsolete. Interestingly, the same condition applies to all other linkage criteria; however, due to space limitations, the proof for all members of the HAC family cannot be specified here.

All in all, at each clustering iteration of HAC, all distances corresponding to one instance (or cluster) become obsolete and can be removed. This potential introduces the possibility of trimming.

Now that the applicability of trimming to HAC is clear, the effect of this technique on computational complexity is investigated. Trimming technique removes just the unnecessary entries from the distance matrix (or any distance representative). For instance, each time the minimum distance between two clusters is found, say i and m in the following matrix, those clusters are put in one cluster (merged). Afterward, old distances of the other cluster (i) are replaced with new ones that are computed by DUS. Since all distances related to one of these clusters (e.g., m) are futile, they can be removed (i.e., by trimming).

$$\begin{pmatrix} d_{11} & d_{12} & . & . & . & d_{1n} \\ d_{21} & d_{22} & . & . & . & d_{2n} \\ . & & d_{im} & & & \\ . & & & . & d_{ij} & \\ . & & & & . & \\ d_{n1} & d_{n2} & . & . & . & d_{nn} \end{pmatrix} \tag{4.14}$$

The trimming process in conventional HAC resembles removing one row and one column related to cluster m from the distance matrix. Therefore, in step k of the clustering process with n data-points, $n-k$ elements are removed from a distance matrix by trimming. The total reduction of computational complexity for finding minimum distances would be $O(n^2)$. However, doing this also injects the costly task of renewing the distance matrix. Here, the worst cases of MFS for two cases are compared: first, vanilla Algorithm 1 in Equation 4.15, and second, MFS with repetitive trimming in Equation 4.16:

$$\sum_{k=1}^{n} n^2 = n^3 \equiv O(n^3) \tag{4.15}$$

$$\begin{aligned} \sum_{k=0}^{n-1}(n-k)^2 + \sum_{k=0}^{n-2}(n-k)^2 &= \\ \frac{n(2n+1)(n+1)}{6} + \frac{n(2n+1)(n+1)}{6} - 1 &= \\ \frac{n(2n+1)(n+1)}{3} - 1 &\equiv \\ O(n^3) \end{aligned} \tag{4.16}$$

It can be seen in Equations 4.15 and 4.16 that trimming per se does not change the asymptotic computational complexity. However, from the practical perspective, in large size real-world dataset clustering, the trimming can significantly increase the clustering speed. this improvement, which applies to all HACs and all data types, is utilized in the current research. It is worth mentioning here that trimming also reduces the computation for distance updating steps (lines 13 and 14 of Algorithm 1), but for the sake of simplicity, it is indicated in Equation 4.16. Next, a more-general idea of updating distances using DUS will be discussed.

Lance-Williams Formula for Cluster Proximity

Recall that each time two clusters are merged a new cluster is formed, so one needs to update the distance between the newly obtained cluster and other clusters.

In each step of HAC, the nearest pair of objects (either clusters or singletons) are found and merged together. Then, the distances between the newly obtained cluster and all other objects are derived according to the selected linkage-criterion. *Lance-Williams update formula* provides a compact update equation that embraces all discussed linkage-criteria. Suppose i and m are clusters that are going to be merged, and l is another cluster whose distance to the merged cluster should be updated. dis_{im} is the distance between clusters i and m, so $dis_{(i\cup m)l}$ would be the distance between clusters $i \cup m$ and l, which can be calculated by the following formula:

$$dis_{(i\cup m)l} = \alpha_i dis_{il} + \alpha_m dis_{ml} + \beta dis_{im} + \gamma|dis_{il} - dis_{ml}| \tag{4.17}$$

where α, β and γ are parameters selected based on a desired linkage-criterion; for example, by plugging in $\alpha_i = 0.5$, $\alpha_m = 0.5$, $\gamma = -0.5$, the update equation for single-linkage [147, 138] is derived. Table 4.2 shows the most well-know HAC methods, their Lance-Williams update parameters, and the coordinate for *cluster representatives*. In some HAC methods such as median, centroid, and Ward HAC, each cluster has a representative and DUS is performed on cluster representatives.

HAC Methods	**Lance-Williams Parameters**	**Coordinate of the Center of the newly merged cluster**
Single-Linkage	$\alpha_i = \alpha_m = 0.5$ $\beta = 0$ $\gamma = -0.5$	NA
Complete-Linkage	$\alpha_i = \alpha_m = 0.5$ $\beta = 0$ $\gamma = 0.5$	NA
Unweighted Average-Linkage	$\alpha_i = \frac{\lvert i\rvert}{\lvert i\rvert+\lvert m\rvert}$ $\alpha_m = \frac{\lvert m\rvert}{\lvert i\rvert+\lvert m\rvert}$ $\beta = 0$ $\gamma = 0$	NA
weighted Average-Linkage	$\alpha_i = \alpha_m = 0.5$ $\beta = 0$ $\gamma = 0$	NA
Median	$\alpha_i = \alpha_m = 0.5$ $\beta = -0.25$ $\gamma = 0$	$\mathbf{C} = \frac{\mathbf{C}_i+\mathbf{C}_m}{2}$
Centroid	$\alpha_i = \frac{\lvert i\rvert}{\lvert i\rvert+\lvert m\rvert}$ $\alpha_m = \frac{\lvert m\rvert}{\lvert i\rvert+\lvert m\rvert}$ $\beta = \frac{\lvert i\rvert\lvert m\rvert}{(\lvert i\rvert+\lvert m\rvert)^2}$ $\gamma = 0$	$\mathbf{C} = \frac{\lvert i\rvert\mathbf{C}_i+\lvert m\rvert\mathbf{C}_m}{\lvert i\rvert+\lvert m\rvert}$
Minimum Variance (Ward)	$\alpha_i = \frac{\lvert i\rvert+\lvert l\rvert}{\lvert i\rvert+\lvert m\rvert+\lvert l\rvert}$ $\alpha_m = \frac{\lvert m\rvert+\lvert l\rvert}{\lvert i\rvert+\lvert m\rvert+\lvert l\rvert}$ $\beta = -\frac{\lvert l\rvert}{\lvert i\rvert+\lvert m\rvert+\lvert l\rvert}$ $\gamma = 0$	$\mathbf{C} = \frac{\lvert i\rvert\mathbf{C}_i+\lvert m\rvert\mathbf{C}_m}{\lvert i\rvert+\lvert m\rvert}$

Table 4.2: DUS Required Parameters Based on Lance-Williams Updating Formula

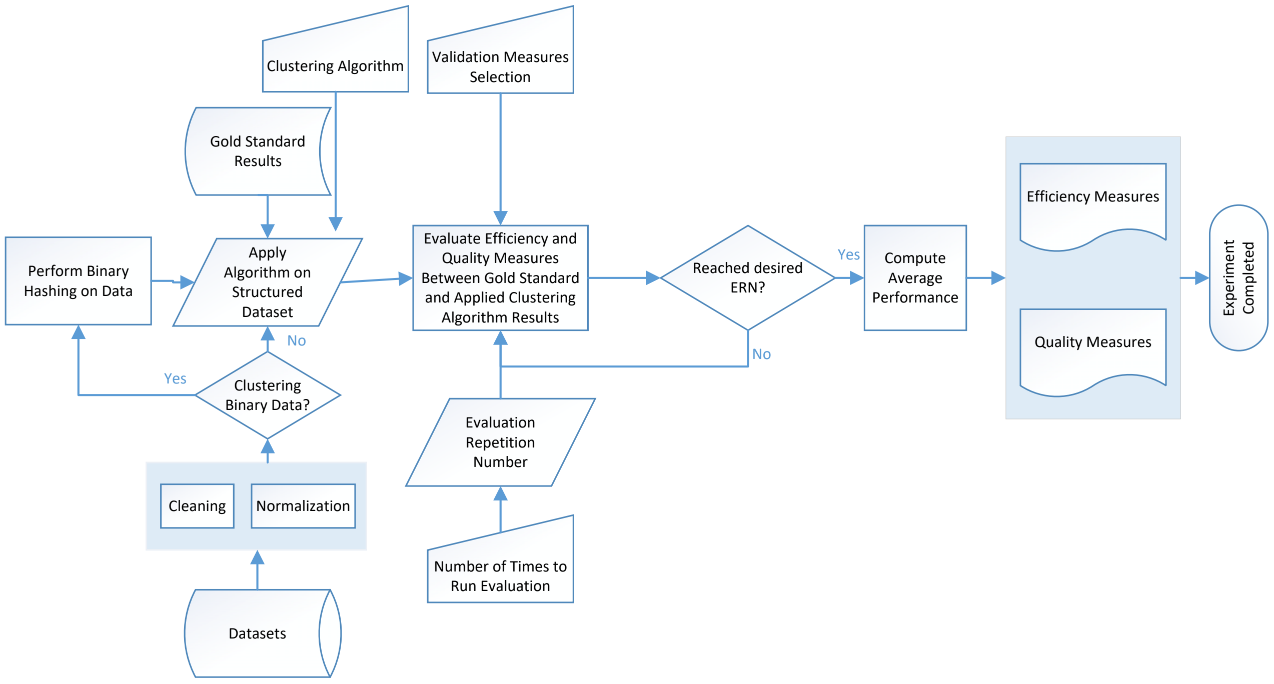
Lance-Williams formula shows that for updating the distances between newly merged clusters (i.e., here $i \cup m$) and other clusters in any HAC method, one merely needs to find the minimum distances between the particular cluster(i.e., l) to each of i and m. Thus, the only information needed to update the distance between any cluster l and the newly formed cluster $i \cup m$ is dis_{il}, dis_{ml} and Lance-Williams parameters (α_i, α_m, β , and γ). All in all, at each clustering iteration of HAC, all distances corresponding to one instance (or cluster) become obsolete and can be removed. This potential introduces the possibility of *trimming*, which is merely removing the unnecessary entries from the distance matrix (or any data structure used for storing distances). With trimming, the computational costs of MFS and DUS can be decreased. However, it also injects the costly task of renewing the distance matrix. Trimming per se does not change the asymptotic computational complexity. However, from the practical perspective, in large size real-world dataset clustering, the trimming can significantly increase the clustering speed.

4.5 Results

The various versions of HAC introduced in Chapter 3 (i.e., DAC) have been discussed and evaluated experimentally to determine which achieves better results for the current research objectives. DAC is compared to alongside conventional HAC algorithms through multiple experiments on all the synthetic and six real-world datasets (Table 4.1). The goal of these experiments is to evaluate the DAC algorithm for its 1) quality measures; 2) performance measures (time and memory costs); and 3) binary code length. This section is divided based on five experiments. In experiment 1, DAC and conventional HAC are compared in $\{0,1\}^d$. Experiment 2 compares DAC and two well-known algorithms: K-means and DBSCAN. Experiment 3 compares the results of DAC in $\{0,1\}^d$ and conventional HAC in $\mathbb{R}^D$. Experiment 4 compares the results of DAC in $\{0,1\}^d$ for various input binary lengths. Experiment 5 compared the results of the adapted DAC for $\mathbb{R}^D$ with non-discretized and discretized distances, where the discretized version includes three different discretization ratios.

The setup for evaluating experimental results is shown in Figure 4.2. *Evaluation Repetition Number* (ERN) is used for multiple runs of clustering algorithms. Additional terms in the figure are clear. The results are assessed based on the evaluation metrics introduced Section 4.1. Table 4.3 describes the abbreviations and acronyms used in the experimental-results tables.

Some points are worth mentioning here. To make the following tables clearer, the best result for each comparison is shown **bold and underlined**. If showing the second-



Clustering Algorithm
Validation Measures Selection
Gold Standard Results
Perform Binary Hashing on Data
Apply Algorithm on Structured Dataset
Evaluate Efficiency and Quality Measures Between Gold Standard and Applied Clustering Algorithm Results
Reached desired ERN?
Yes
No
Compute Average Performance
Efficiency Measures
Quality Measures
Experiment Completed
No
Yes
Clustering Binary Data?
Evaluation Repetition Number
Cleaning
Normalization
Number of Times to Run Evaluation
Datasets

Figure 4.2: Setup of Experiment

best result is helpful, it is in **bold**. The mean values of the results are shown. As the starting points can affect clustering results, multiple runs of clustering algorithms on shuffled datasets have been performed, and the means of the results are calculated. Running time is the summation of the normalization, dimensionality reduction, hashing, distance computation, and clustering times.

4.5.1 Experiment 1

This section describes the comparison between DAC (discussed in Chapter 3) and conventional HAC in $\{0,1\}^d$. DAC utilizes two arrays of linked-lists instead of a distance matrix for storing calculated mutual distances between all data-points. By employing these combinatorial data structures, naturally sorted distances can be built, thus eliminating the pairwise nearest-neighbour search that is the most costly part of HAC.

This experiment is conducted on all six datasets: CIFAR10, TEXMEX, MNIST, EColi, IRIS, and Image segment, introduced in Table 4.1. After running the proposed and conventional algorithms on these datasets (which also returns the running time), the results

Acronym	Description
DBSCAN	One of the most well-known algorithms, and works based on density. It defines clusters by finding high density regions which are separated by low density regions.
Eps	The minimum distance between points as an input parameter for DBSCAN algorithm. If the distance between two points is less than or equal to Eps, these points are considered to be neighbors.
HamCon	Improved conventional HAC in Hamming space
HamLL	Linked-Lists version of hierarchical agglomerative clustering algorithm (i.e., DAC) in Hamming space
ImpEuc	Improved conventional HAC in Euclidean space
ImSegment	Image segmentation dataset (see Table 4.1)
K	Number of clusters as an input parameter for K-means algorithm
MPts	Minimum number of points as an input parameter for DBSCAN algorithm. This number of points needs to be within the Eps; otherwise, a newly evaluated point is ruled out momentarily and will be evaluated later and perhaps considered to be noise.

Table 4.3: Abbreviations and Acronyms Used in Experimental Tables

of the algorithms are also evaluated by performance measures (Rand index, adjusted Rand index, and purity) and running time, as discussed in 4.1 and 4.1.1. In Table 4.4, the value of each performance measure is interpreted in order to gain a clear view of the quality of the algorithms.

The following tables show the results of the performance comparison in experiment Number 1 for various measures mentioned in Table 4.4 utilized and used to evaluate the algorithms. Each value in this table represents the mean of 5 runs per algorithm on the dataset. At each iteration, data is shuffled to remove any bias.

The running times in Table 4.3 show that, in all cases, DAC performs clustering faster than the improved conventional HAC. Datasets of various sizes and dimensionalities have been selected. This table demonstrates that a different number of records in a dataset can drastically change the clustering time. The power of the proposed clustering approach is revealed when the size of the dataset or its dimensionality grows. The clustering time for large-size datasets is by orders of magnitude faster than the time of improved conventional HAC. When the number of records in a dataset, n, grows, the distance computation part

Measure	Performance Indication	High/Low
Purity	The higher the value of purity, the higher the quality and effectiveness of the algorithm	+
RI	The higher the value of RI, the higher the quality and effectiveness of the algorithm	+
ARI	The higher the value of ARI the higher the quality and effectiveness of the algorithm	+
Time	The lower the value of time, the higher the efficiency and the faster the algorithm	-

Table 4.4: Performance Indications

for any exact HAC (with the computational complexity of $O(n^2)$) becomes costlier, but this cost increment is inevitable. Finding the minimum distance between $n(n-1)/2$ distances for n times (starting from singleton clusters to one cluster) becomes the most costly part, with the computational complexity of $O(n^3)$. This part has been removed by applying the proposed algorithm, so the growth of the dataset has less effect on resources and running-time.

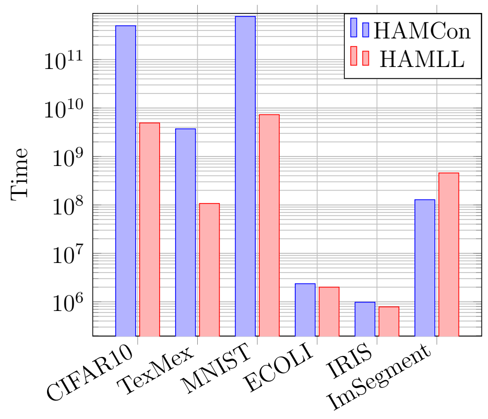
Although in some cases, the performance results for DAC are slightly lower than those for conventional HAC, the table also shows that, in most cases, DAC outputs better results, and the small performance measure differences are negligible. The output clusters are also compact and well-separated compared to these with conventional HAC in $\{0,1\}^d$. The Adjusted Rand index, which is one of the most clarifying quality measures in the current experiments, in all datasets except TEXMEX, shows better clustering quality for DAC.

This experiment concludes that DAC performes regarding the quality of clustering, which shows clusters' separation and compactness. The more-critical performance measure is the efficiency of algorithms, which shows that DAC performs faster than conventional algorithms. Therefore, DAC is superior in this study on binary datasets because of its better performance measures including all efficiency and effectiveness measures.

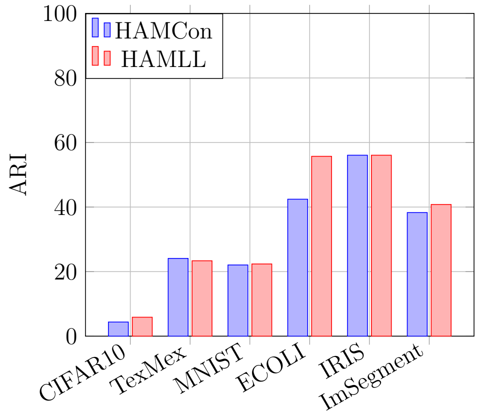
The next experiment compares of DAC with two well-known clustering algorithms: k-means and DBSCAN.

4.5.2 Experiment 2

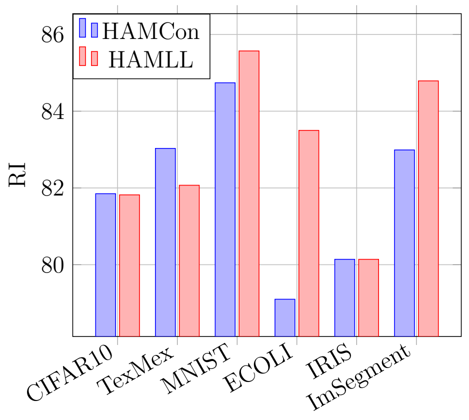
This section presents the comparison between DAC (discussed in Chapter 3) and two well-known non-hierarchical clustering algorithms: DBSCAN and k-means. As mentioned, all



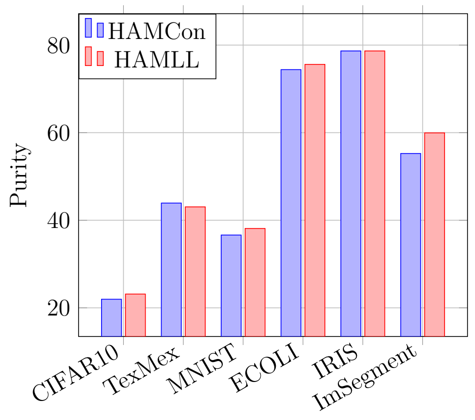
(a) Time Comparison



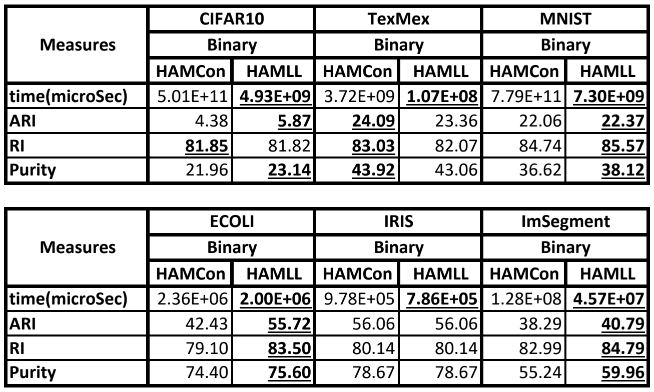
(b) ARI Comparison



(c) RI Comparison



(d) Purity Comparison



Measures	CIFAR10		TexMex		MNIST	
	Binary		Binary		Binary	
	HAMCon	HAMLL	HAMCon	HAMLL	HAMCon	HAMLL
time(microSec)	5.01E+11	**4.93E+09**	3.72E+09	**1.07E+08**	7.79E+11	**7.30E+09**
ARI	4.38	**5.87**	**24.09**	23.36	22.06	**22.37**
RI	**81.85**	81.82	**83.03**	82.07	84.74	**85.57**
Purity	21.96	**23.14**	**43.92**	43.06	36.62	**38.12**

Measures	ECOLI		IRIS		ImSegment	
	Binary		Binary		Binary	
	HAMCon	HAMLL	HAMCon	HAMLL	HAMCon	HAMLL
time(microSec)	2.36E+06	**2.00E+06**	9.78E+05	**7.86E+05**	1.28E+08	**4.57E+07**
ARI	42.43	**55.72**	56.06	56.06	38.29	**40.79**
RI	79.10	**83.50**	80.14	80.14	82.99	**84.79**
Purity	74.40	**75.60**	78.67	78.67	55.24	**59.96**

(e) HAC VS. DAC

Figure 4.3: Comparison Between the Improved Conventional HAC and DAC

datasets are not initially binary, so these clustering algorithms have been applied on a transformed binary version of datasets, and obviously, the lengths of the input binary code for all the algorithms are the same.

Table 4.5 lists the parameters set up for each dataset and the two algorithms, k-means and DBSCAN. K-means only needs one parameter which is k (i.e., the number of clusters as input). DBSCAN has two input parameters: (i) the minimum number of points, and (ii) the radius. Both algorithms perform clustering during one run only. However, finding an optimum Eps can be tricky and may need multiple runs. These pre-clustering runs are not considered for running time calculation, and only the mean of 5 final iterations is used.

	Dataset-Related Input Parameters		
Dataset Name	**MPts**	**Eps**	**K**
Iris	4	0.25	3
Ecoli	4	0.4	5
Image Segmentation	4	0.5	7
CIFAR-10	4	0.65	10
TEXMEX-ANNSIFT10K	4	0.45	10
MNIST	4	0.25	10

Table 4.5: Dataset Specifications

Table 4.4 show the results of Experiment 2. These tables represent the evaluation of the aforementioned performance measures, to find which algorithm performs better. Each value in this table represents the mean of 5 runs per algorithm on the dataset. At each iteration, data is shuffled to remove any bias. K-means algorithm has the best running time for most datasets, which is theoretically provable. It can be seen in Chapters 2 and 3 that the computational complexity of k-means ($O(tknd)$) is lower than that of both DBSCAN and the proposed approach (both $O(n^2d)$). In some cases, DBSCAN algorithm performs clustering faster than k-means. However, the performance measures of DBSCAN are the lowest in all datasets, which make this algorithm erroneous in Hamming space. The performance measures for the k-means algorithm are slightly better than those of the proposed algorithm, and for running time, k-means outperforms DAC in large-scale datasets. However, being deterministic and nonparametric are two major benefits of HAC and subsequently the proposed approach.

The effect of hashing on clustering algorithms is not the goal of this thesis. However, results of the other experiments (not discussed here) show that applying k-means on binary codes, with similar code lengths to those in this experiment, outperforms k-means in

Euclidean space by drastically lower running time and less than one percent difference in ARI.

The next experiment provides the results for DAC in $\{0,1\}^d$ and conventional HAC in $\mathbb{R}^D$.

4.5.3 Experiment 3

Experiment 3 compares results for DAC in $\{0,1\}^d$ and conventional HAC in $\mathbb{R}^D$. It is shown that one way to utilize the advantages of HAC without suffering too much from its computational costs is by transferring data from $\mathbb{R}^D$ to $\{0,1\}^d$ and employing DAC on binary codes.

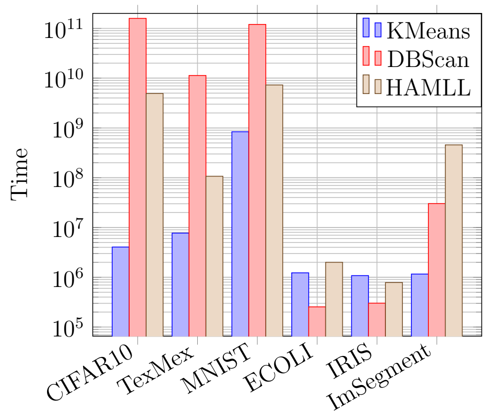
Therefore, in this experiment, the conventional HAC in $\mathbb{R}^D$ has been utilized and its performance measures extracted (as summarized in Table 4.4). Then DAC is employed in transferred $\{0,1\}^d$ with different binary code lengths and the results evaluated. There is no input parameter for conventional HAC in $\mathbb{R}^D$, but one can consider binary code length as the input parameter for transferring data to binary codes as the patch to the proposed approach for $\mathbb{R}^D$ utilization. Thus, binary code lengths (in parenthesis) are shown beside the approaches in the following tables.

Table 4.5 show the results of Experiment 3, representing the evaluation of performance measures to find which algorithm performs better.

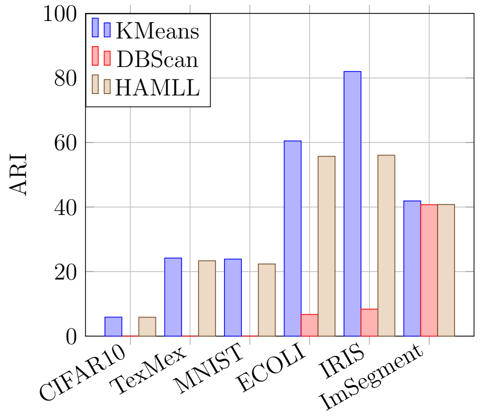
The values are the result of running algorithms on the mentioned dataset. The running time of improved conventional HAC in $\mathbb{R}^D$ is the summation of the Euclidean distance computation times between all pair of points and HAC times. Recall that the Euclidean distance between two points $\mathbf{p}, \mathbf{q}$ is :

$$\begin{aligned} d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) &= \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2} \\ &= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}. \end{aligned} \tag{4.18}$$

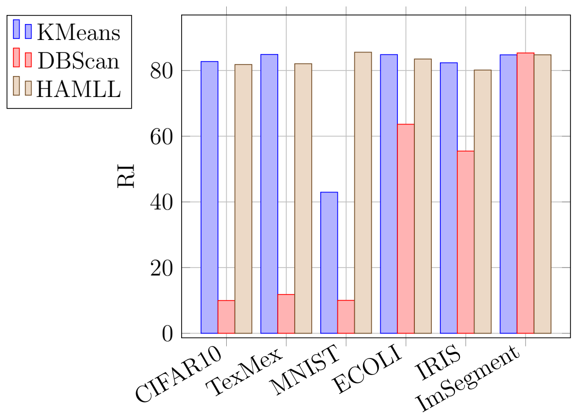
The only difference between conventional HAC in $\mathbb{R}^D$ and conventional HAC in $\{0,1\}^d$ is the elimination of this computation. In $\{0,1\}^d$, one can simply use a lookup-table (corresponding to binary code length) and extract distances without any computation. However, as noted, this research stepped forward and attacked the most costly part of conventional HAC. Therefore, the approach has a twofold improvement: first, Euclidean



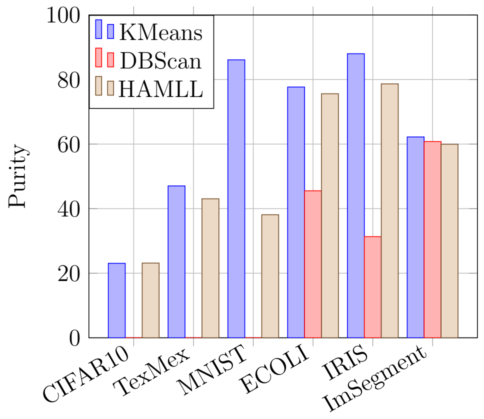
(a) Time Comparison



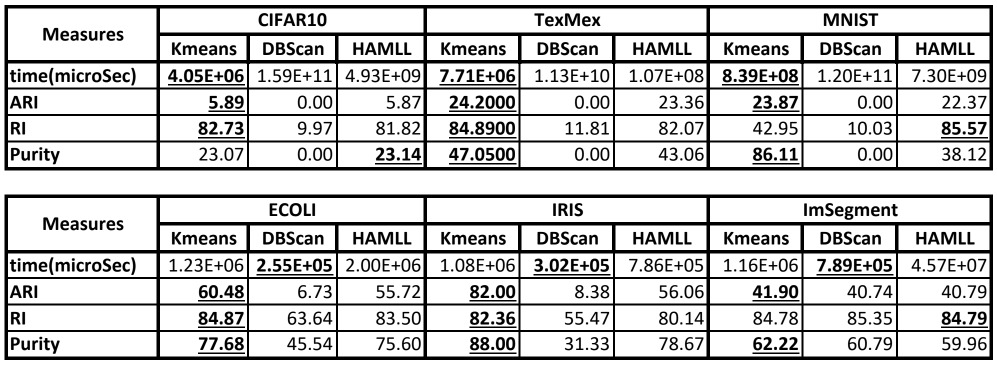
(b) ARI Comparison



(c) RI Comparison



(d) Purity Comparison



Measures	CIFAR10			TexMex			MNIST		
	Kmeans	DBScan	HAMLL	Kmeans	DBScan	HAMLL	Kmeans	DBScan	HAMLL
time(microSec)	**4.05E+06**	1.59E+11	4.93E+09	**7.71E+06**	1.13E+10	1.07E+08	**8.39E+08**	1.20E+11	7.30E+09
ARI	**5.89**	0.00	5.87	**24.2000**	0.00	23.36	**23.87**	0.00	22.37
RI	**82.73**	9.97	81.82	**84.8900**	11.81	82.07	42.95	10.03	**85.57**
Purity	23.07	0.00	**23.14**	**47.0500**	0.00	43.06	**86.11**	0.00	38.12

Measures	ECOLI			IRIS			ImSegment		
	Kmeans	DBScan	HAMLL	Kmeans	DBScan	HAMLL	Kmeans	DBScan	HAMLL
time(microSec)	1.23E+06	**2.55E+05**	2.00E+06	1.08E+06	**3.02E+05**	7.86E+05	1.16E+06	**7.89E+05**	4.57E+07
ARI	**60.48**	6.73	55.72	**82.00**	8.38	56.06	**41.90**	40.74	40.79
RI	**84.87**	63.64	83.50	**82.36**	55.47	80.14	84.78	85.35	**84.79**
Purity	**77.68**	45.54	75.60	**88.00**	31.33	78.67	**62.22**	60.79	59.96

(e) KMeans VS. DBScan VS. DAC

Figure 4.4: Comparison Between two Well-known Clustering Algorithms and DAC

distance elimination, and second, removing pairwise nearest neighbor searching from the distance matrix.

Running times in Table 4.5 show that the proposed algorithm is remarkably faster than improved conventional HAC. Obviously, in many dimensionality-reduction or feature-compression techniques (including LSH) some part of original data is considered as the loss of the transformation. In most datasets, performance measures for improved conventional HAC in $\mathbb{R}^D$ are higher than that for DAC, in $\{0,1\}^d$ which is the result of data loss (the trade-off between accuracy and time).

An adjusted Rand index gives negative points to wrong clustering assignments, making it different from RI. The maximum purity difference between compared approaches within all datasets is around 12 percent, which is also the result of data loss in the hashing process.

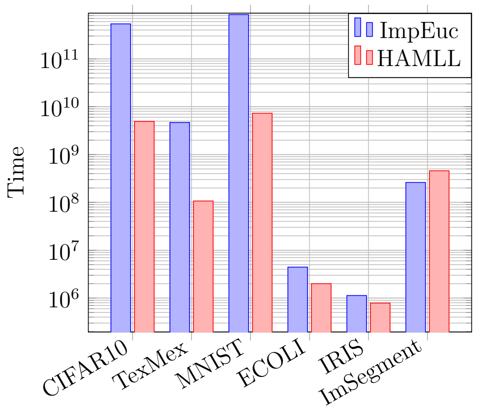
This experiment demonstrates the expansion of DAC to $\mathbb{R}^D$. Experiment 3 shows that by transferring data to $\{0,1\}^d$ and applying DAC, one can have a reasonably accurate algorithm for agglomerative clustering in $\mathbb{R}^D$. Most importantly, DAC performs clustering significantly faster than HAC algorithms, even with the time it takes to transfer data to $\{0,1\}^d$. The performance measure loss is the result of the binary hashing process and depends on the nature of datasets. All being said, one can try reduced and shuffled versions of datasets and perform conventional HAC or DAC on small datasets, and then, based on performance measures, decide to stick with the conventional approach or DAC.

In this experiment, the effect of information loss on clustering results can be seen. In the next section, different binary code lengths are utilized to demonstrate their effect on clustering performance measures.

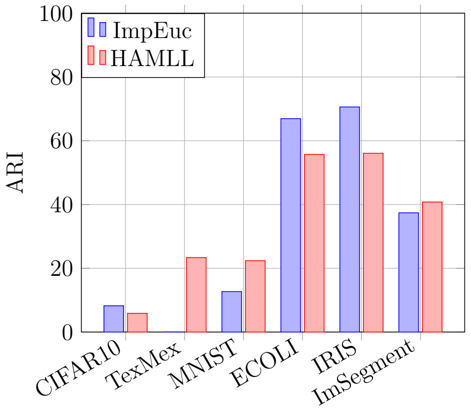
4.5.4 Experiment 4

This section compares the various lengths of binary codes for DAC that have been discussed in Chapter 3.

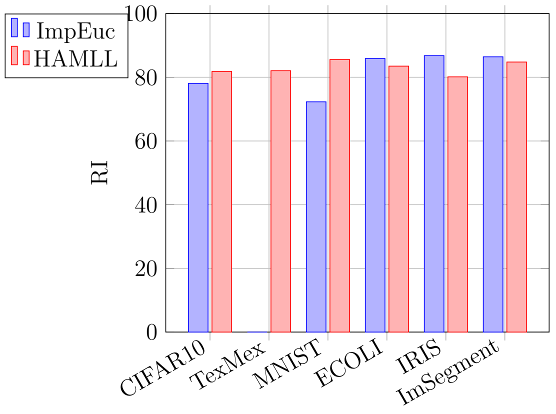
This experiment is conducted on three large datasets where the original dimensions (before FC or DR) are high enough for long binary code target. The introductions in Table 4.1 clarifies why. The first step is transferring data to $\{0,1\}^d$ by applying LSH with different binary code lengths then running DAC on these transformed datasets. Performance measures evaluate the results of the algorithms. The interpretation of each performance measure can be found in Table 4.4, providing a clear view of the quality of the algorithms and the effects of binary code length on the algorithm. Table 4.6 shows the results of performance comparison in experiment no. 4 for the various measures mentioned in Table 4.4



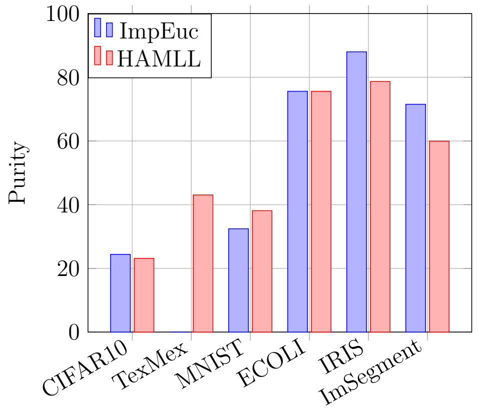
(a) Time Comparison



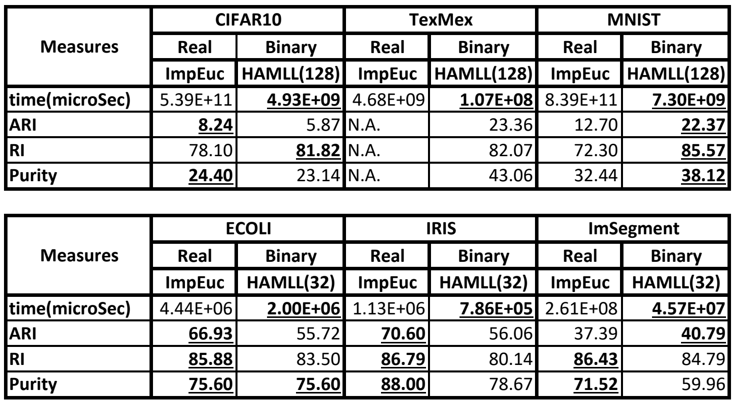
(b) ARI Comparison



(c) RI Comparison



(d) Purity Comparison



Measures	CIFAR10		TexMex		MNIST	
	Real	Binary	Real	Binary	Real	Binary
	ImpEuc	HAMLL(128)	ImpEuc	HAMLL(128)	ImpEuc	HAMLL(128)
time(microSec)	5.39E+11	**4.93E+09**	4.68E+09	**1.07E+08**	8.39E+11	**7.30E+09**
ARI	**8.24**	5.87	N.A.	23.36	12.70	**22.37**
RI	78.10	**81.82**	N.A.	82.07	72.30	**85.57**
Purity	**24.40**	23.14	N.A.	43.06	32.44	**38.12**

Measures	ECOLI		IRIS		ImSegment	
	Real	Binary	Real	Binary	Real	Binary
	ImpEuc	HAMLL(32)	ImpEuc	HAMLL(32)	ImpEuc	HAMLL(32)
time(microSec)	4.44E+06	**2.00E+06**	1.13E+06	**7.86E+05**	2.61E+08	**4.57E+07**
ARI	**66.93**	55.72	**70.60**	56.06	37.39	**40.79**
RI	**85.88**	83.50	**86.79**	80.14	**86.43**	84.79
Purity	**75.60**	**75.60**	**88.00**	78.67	**71.52**	59.96

(e) HAC in Euclidean Space VS. DAC

Figure 4.5: Comparison of Improved Conventional HAC in $\mathbb{R}^D$ and DAC in $\{0, 1\}^d$

and used for evaluating of the algorithms. The values are the result of running algorithms on the dataset. The running times in these three versions are close, so they are not be mentioned here.

It can be seen in Table 4.4 that, in almost all cases, more-extended binary codes result in better performance measures. The result of improved conventional HAC is brought in to show that the dependency of the results on the length of binary codes is not limited to DAC. Apparently, the longer the binary codes, the higher the percentage of original data preserved. Although the running time is slightly different for each dataset-approach, as shown in Table 4.3, there is a huge difference in running time between the two approaches for a specific dataset. Therefore, if less than 0.1% difference in performance measure is not an issue for the user, DAC is recommended due to its order-of-magnitude-smaller running time, especially on large-scale datasets.

Next, applying the proposed approach to $\mathbb{R}^D$ datasets, without transferring data into $\{0,1\}^d$ before clustering, is discussed.

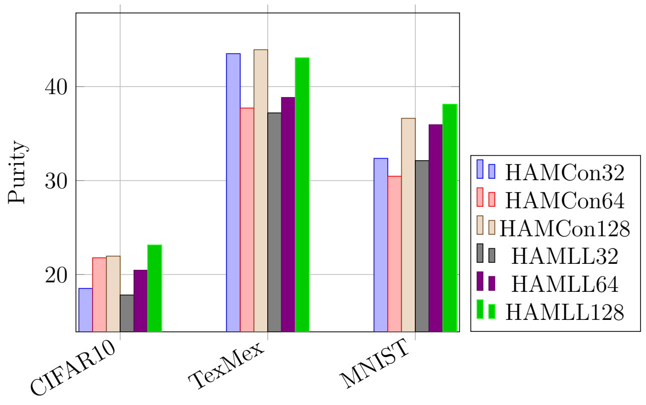
4.5.5 Experiment 5

To this point, the clustering in $\{0,1\}^d$ was conducted so that in some datasets (which are not naturally binary) the result of binary hashing. Although LSH does not take up a big portion of the whole running time, it is worth thinking about eliminating this step. However, talking about the idea and the motivations is essential. The simplified calculation of computational complexity for conventional HAC is the summation of two parts: first, distance computation $O(n^2d)$, where n is the number of records and d is the dimension, and second, clustering with computational complexity of $O(n^3)$ (refer to Chapter 3). Thus, the total computational complexity is $O(n^3)$.

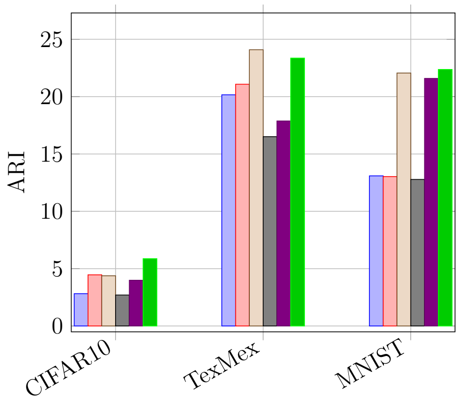
By transferring data to $\{0,1\}^d$ with lower dimensions, and utilizing lookup-tables for distance calculation, the computational complexity of the first step becomes $O(n^2)$. For the second step, an algorithm was proposed to eliminate the minimum finding step within the distance matrix that decreased its computational complexity to $O(n^2)$. The cost of SimHash LSH is negligible compared to $O(n^2)$, making the total complexity $O(n^2)$.

Now, given a dataset with n number of records and D dimensions in $\mathbb{R}^D$, if one can use DAC on calculated Euclidean distances, the total computational complexity would be $O(n^2)+O(n^2D)$ or $O(n^2D)$. In this case, if $D << n$, this computational complexity would be far less than $O(n^3)$, which is the case for conventional HAC in $\mathbb{R}^D$.

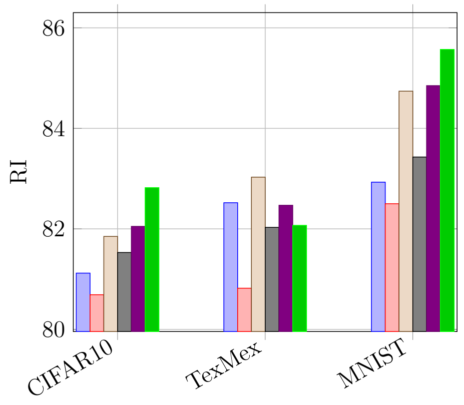
Consider a distance space derived from a Euclidean space with n number of points and d dimensions ($\mathbb{R}^d$). The distance space will be $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} : x \geq 0\}$. The difference



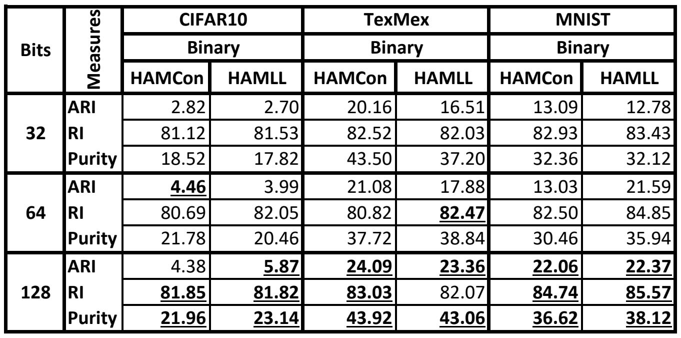
(a) Purity Comparison



(b) ARI Comparison



(c) RI Comparison



Bits	Measures	CIFAR10		TexMex		MNIST	
		Binary		Binary		Binary	
		HAMCon	HAMLL	HAMCon	HAMLL	HAMCon	HAMLL
32	ARI	2.82	2.70	20.16	16.51	13.09	12.78
	RI	81.12	81.53	82.52	82.03	82.93	83.43
	Purity	18.52	17.82	43.50	37.20	32.36	32.12
64	ARI	**4.46**	3.99	21.08	17.88	13.03	21.59
	RI	80.69	82.05	80.82	**82.47**	82.50	84.85
	Purity	21.78	20.46	37.72	38.84	30.46	35.94
128	ARI	4.38	**5.87**	**24.09**	**23.36**	**22.06**	**22.37**
	RI	**81.85**	**81.82**	**83.03**	82.07	**84.74**	**85.57**
	Purity	**21.96**	**23.14**	**43.92**	**43.06**	**36.62**	**38.12**

(d) HAC in Euclidean Space VS. DAC

Figure 4.6: Comparison of Various Binary Lengths in DAC

between $\mathbb{R}_{\geq 0}$ and $\mathbb{N}$ is the continuous region ($\mathbb{R}_{\geq 0} - \mathbb{N}$) that prevents applying DAC in Euclidean space problems:

$$\mathbb{R}_{\geq 0} = \{0\} \cup (0,1) \cup \{1\} \cup (1,\infty) \tag{4.19}$$

$$\mathbb{N} = \{0\} \cup \{1\} \cup \{2\}... \tag{4.20}$$

$$\mathbb{R}_{\geq 0} - \mathbb{N} = (0,1) \cup (1,2) \cup (2,3)... \tag{4.21}$$

The goal here is to divide each continuous region (from $\mathbb{R}_{\geq 0} - \mathbb{N}$) into k bins. k is called the *Discretization Ratio (DIR)*. There are several possibilities for making these bins, but it is preferable to multiply each continuous region by k and then apply the nearest integer function on this multiplication result. The nearest integer function for any real number x is denoted by $\lfloor x \rceil$. This approach makes DAC applicable to these integer distances.

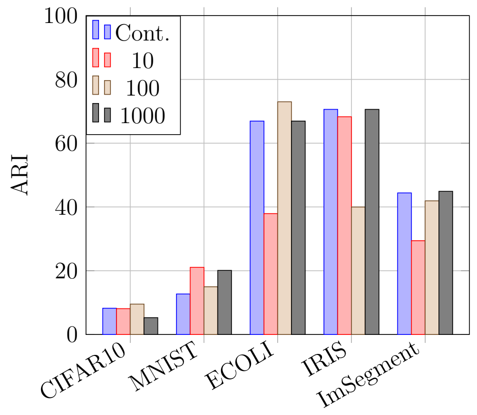
This part shows the comparison between the various DIRs for DAC that have been discussed in Chapter 3. This experiment was conducted on the five datasets introduced in Table 4.1. Performance measures evaluated the results of the algorithms. The interpretation of each performance measure found in Table 4.4 provide a clear view of the quality of the algorithms and effect of DIR on the algorithm. Table 4.7 shows the results of a performance comparison in Experiment 5 for the various measures mentioned in Table 4.4 and utilized for evaluating of the algorithms.

This comparison is between conventional HAC on continuous distance space and DAC on discretized distance space with three different DIRs (k =10, 100, and 1000). For all datasets, the maximum ARI can be achieved by a discretization approach and DAC. Also, for almost all cases, the highest purities and RIs are within this range of discretization.

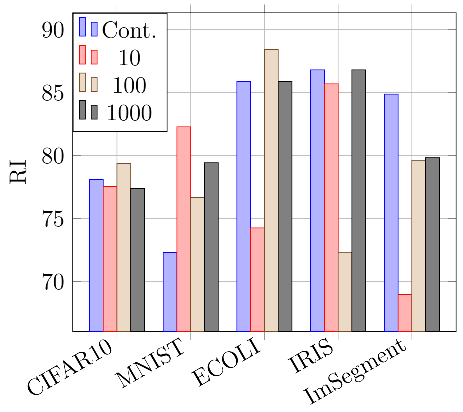
The combination of this experiment and Experiment 3 shows that a discretization approach without hashing can be considered as an alternative, especially when both running time and performance measures are significant concerns to the user.

4.5.6 Discussion

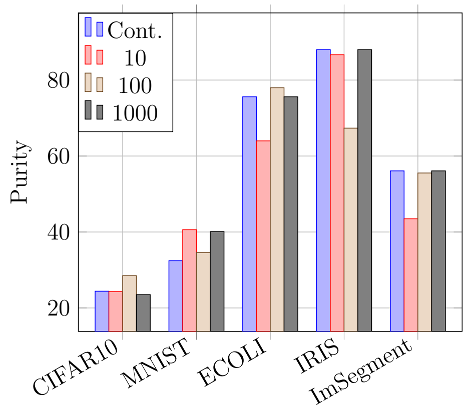
Several experiments have been conducted to evaluate the proposed algorithm that incorporates a new version of storing distances between binary data, such that sorted distances are accessible, and random access to instances is also available. The discrete nature of binary data is utilize to combine array and linked-list data structures as DAC. It is demonstrated that the proposed algorithm performs well in many cases. Here, the relevant answers for each of the research questions are provided.



(a) ARI Comparison



(b) RI Comparison



(c) Purity Comparison

Measures	discretization ratio	CIFAR10	MNIST	ECOLI	IRIS	ImSegment
		Real	Real	Real	Real	Real
		Euclidean	Euclidean	Euclidean	Euclidean	Euclidean
ARI	**Continuous**	8.24	12.70	66.93	**70.60**	44.40
	10	8.11	**21.06**	37.92	68.28	29.42
	100	**9.53**	14.97	**72.98**	39.98	41.91
	1000	5.28	20.1	66.93	**70.6**	**44.9**
RI	**Continuous**	78.10	72.30	**85.88**	**86.79**	**84.78**
	10	77.54	**82.27**	74.25	85.68	68.96
	100	**79.37**	76.66	88.40	72.32	79.62
	1000	77.37	79.42	85.87	**86.79**	79.82
Purity	**Continuous**	24.40	32.44	**75.60**	**88.00**	**56.09**
	10	24.30	**40.60**	63.99	86.67	43.48
	100	**28.50**	34.6	77.98	67.33	55.53
	1000	23.5	40.1	**75.6**	**88**	**56.09**

(d) DAC With Different Discretization Ratio

Figure 4.7: Comparison of Various Discretization Ratios in DAC for $\mathbb{R}^D$

RQ1: Given a compact discrete version of the data or its inter-instant distances, is it possible to develop a data structure and algorithm in a way that minimizes the pairwise nearest neighbour searches of hierarchical agglomerative clustering algorithm?

Yes, a data structure and algorithm named DAC has been implemented that includes two arrays of linked lists, ADLL and AALL. From the theoretical perspective, DAC algorithms can reduce the computational cost of hierarchical agglomerative clustering from cubic to quadratic, matching the known lower bounds of HAC. The empirical results also indicate that DAC can achieve several orders of magnitude speed without losing clustering quality.

RQ2: If there is a data structure and algorithm, is there any specification for the type of data?

To implement the proposed algorithm, the discrete nature of binary codes has been utilized. The proposed data structure and algorithm DAC-BALL are applicable to any discrete code with a bounded length. In the case of binary codes, the best code length (for applying DAC-BALL) equals the processor's datapath widths, integer size, and memory address widths. As an example, in order to compute Hamming distances on a 64-bit processor (bit-wise computation at hardware level), the best length of binary code would be 64 bits.

RQ3: Does the space complexity of HAC not increase prohibitively with such a data structure?

The space complexity of the proposed algorithm (DAC) and its corresponding data structure (BALL) is $O(n^2)$, which is equal (with higher constants) to conventional HAC. Both algorithms need to store the distances in memory, an inevitable part of exact agglomerative clustering algorithms.

The performance of DAC was measured regarding efficiency and effectiveness measures (briefly performance measures). Table 4.6 summarizes the experimental results.

Next (Table 4.7), the clustering features fulfilled by DAC are briefly pointed out to clarify DAC's current potency and potential future improvements. A plus sign (+) means that the specific feature is fulfilled, and a minus sign (−) shows that the algorithm is weak.

Experiment	**Summary of Experiment Results**
No. 1	The proposed algorithm is remarkably faster than improved conventional HAC, without loss of clustering quality measures.
No. 2	The performance measures of DAC are at least second best compared to k-means and DBSCAN. K-means' running time is also best. DAC is deterministic and parameter-free in contrast to DBSCAN and k-means
No. 3	The SimHash LSH adds information loss to the problem. Thus, the quality of DAC in $\{0,1\}^d$ is lower than HACs in $\mathbb{R}^D$; however, with a remarkably lower running time
No. 4	The longer the binary codes, the higher the performance measures.
No. 5	Discretization approach combined with DAC can be considered as an alternative when both running time and performance measures are major concerns.

Table 4.6: Summary of Experiment Results

Clustering Feature	**DAC**	**Conventional HAC**
Handling Various Clustering Shapes	+	+
Exact Algorithm	+ (Limited to Finite Discrete Spaces)	+
Handling Various Similarity Measures	+ (Limited to Finite Discrete Spaces)	+
Handling High Dimensionality	+	-
Handling Big Data	+	-
Parameter Free or Automatic Parameter Discovery	+ (Limited to Finite Discrete Spaces)	+
Handling Various Neighborhood Densities	+	+

Table 4.7: Features Captured by DAC vs Conventional HAC

4.6 Summary

In this chapter, the experimental results for DAC and its corresponding data structure (BALL) in $\{0,1\}^d$ have been presented. Also, the efficiency and accuracy of the proposed algorithm and conventional HAC are compared based on evaluation measures that were presented in Section 4.1. The efficiency and accuracy of the algorithm confirm its total performance. The efficiency is related to the resources used to run a clustering algorithm, such as running time. The accuracy of the algorithm is based on the quality of clustering results. Section 4.3, has described the datasets that are involved in the experiments. Section 4.5 shows the results of five experiments on the datasets, evaluated on six real-world datasets and ten synthetic datasets. The evaluation results on the real-world datasets are also presented. Efficiency and accuracy measures have assessed the results of the experiments. From the theoretical perspective, DAC algorithm reduces the computational complexity of exact agglomerative clustering from cubic to quadratic. Just as importantly, the empirical evaluations, according to the metrics (measures) used, on real-world large-scale datasets, show several orders of magnitude speedup in comparison with conventional techniques.

Chapter 5

Conclusion and Future Work

The main research contributions and limitations are reviewed in this chapter, followed by a discussion on potential future research directions.

This thesis has proposed a new version of *Hierarchical Agglomerative Clustering (HAC)* have been proposed that combines arrays and linked-lists in Hamming space: *Discretized Agglomerative Clustering (DAC)*. The algorithm achieves the theoretical lower bound for an agglomerative clustering family of algorithms by taking advantage of the discrete nature of binary data.

The proposed algorithm has been evaluated experimentally by comparing it with conventional HAC and also with well-known algorithms such as K-means or DBSCAN. Then all these algorithms have been validated on various real-world and synthetic datasets. The accuracy and efficiency metrics such as purity, Rand index, and running time were the primary concentration of the experimental-evaluation process. The accuracy metrics used in this study (i.e. purity, Rand index, and adjusted Rand index) were selected from external evaluation measures that take advantage of having ground truth. In the absence of ground truth, using internal evaluation measures that show the compactness or separateness of clusters (refer to Appendix C) is also applicable. As mentioned in Chapter 4, HAC algorithms are not parameter sensitive, and so the evaluation has focused on cluster shapes, memory usage, and running time.

The experimental analysis shows that DAC-BALL improves the computational cost and efficiency of the conventional HAC algorithms without losing much accuracy in the (clustering) results. It has also been shown that by transferring data from $\mathbb{R}^D$ to $\{0,1\}^d$, DAC becomes much more efficient than HAC, in $\mathbb{R}^D$, without losing much accuracy. This algorithm removes the pairwise nearest neighbor search step (MFS) in HAC algorithms by merging this step into the DCS. To the best of the reseachers' knowledge, no other exact agglomerative clustering algorithm both covers all members of the HAC family and has the worst-case computational complexity of quadratic. Using DAC is applicable to many other categorical data clustering applications where the distances are discrete and bounded.

In summary, the proposed algorithm clusters binary data based on pairwise distances by exploiting the discrete and bounded nature of distances. It removes the need for a pairwise nearest neighbour search by storing distances in a sorted consistent manner. It deletes distances corresponding to a specific cluster in constant time, which makes the algorithm applicable to all members of the HAC family. This algorithm can be utilized not only on categorical data but also in $\mathbb{R}^D$ space by mapping it to $\{0,1\}^d$. Accordingly, it is concluded that DAC, which uses a hybrid data structure (i.e., BALL), is a scalable exact clustering approach for many applications.

5.1 Research Contributions

This thesis contributes to the field of clustering as a part of data mining by proposing the DAC and its various useful characteristics:

- Discrete distances are adapted to enable HAC to perform the following:
  - It merges clusters (points) considering closest pairs without a pairwise nearest neighbour search.
  - It updates all corresponding distances while merging two clusters without violating the consistency of stored sorted distances.
  - It enables random access to the distance between two clusters.
- A combination of data structures is introduced that performs the following:
  - It enables DAC to execute exact HAC on binary data with the lowest computational complexity that the researchers are aware of.
  - It enables DAC to perform exact HAC on any bounded discrete space at the lowest computational cost.
  - Transferring $\mathbb{R}^D$ to $\{0,1\}^d$ enables efficient approximate HAC in $\mathbb{R}^D$ with small accuracy loss, and is drastically faster than conventional HAC.
  - It creates the possibility of approximate HAC in $\mathbb{R}^D$ by discretizing the space such that distances are finite integers. This approach needs no transformation to $\{0,1\}^d$.

5.2 Threats to Validity

It is possible to identify limitations of the DAC algorithm by focusing on certain clustering features of the DAC algorithm not yet achieved. These features can be extracted from optimal algorithms. The features can also be identified through a literature review of clustering algorithms or even a wider range of data mining algorithms. These features can also be called *requirements*. DAC was evaluated in Chapter 4 based on criteria presented as accuracy and efficiency metrics. Here, those metrics are represented in Table 5.1 as features of the clustering algorithm. If DAC gains a feature, + is used and − vice-versa.

It is worth mentioning that one should distinguish between the inherited aspects of DAC and specific characteristics of DAC. If the conventional HAC is considered as a base class (superclass), DAC is a subclass that inherits some of its behaviour from its superclass.

Clustering Feature	**DAC**	**Conventional HAC**
Handling Various Clustering Shapes	+	+
Exact Algorithm	+ (Limited to Finite Discrete Spaces)	+
Handling Various Similarity Measures	+ (Limited to Finite Discrete Spaces)	+
Handling High Dimensionality	+	-
Handling Big Data	+	-
Parameter Free or Automatic Parameter Discovery	+ (Limited to Finite Discrete Spaces)	+
Handling Various Neighborhood Densities	+	+

Table 5.1: Features Captured by DAC vs Conventional HAC

As shown in Table 5.1, three features in DAC are satisfied, but limitations remain. For instance, DAC performs exact clustering only on discrete finite spaces, and its execution in $\mathbb{R}^D$ is approximate, although evaluation experiments have shown satisfactory results. The same reasoning is applicable for "Handling Various Similarity Measures". If DAC is executed on $\{0,1\}^d$, it is evident that one is limited to using $\{0,1\}^d$ similarity measures. This mapping of $\mathbb{R}^D$ data to $\{0,1\}^d$ is considered to be feature extraction, and thus it needs the number of desired output features as the input parameter. The need for this parameter is another limitation of DAC in $\mathbb{R}^D$ problems.

One apparent disadvantage of HAC is its high computational complexity, and even though the computational complexity cost has been decreased by the proposed approach to close to its lower bounds (i.e. $O(n^2)$), it is still costly compared to some clustering algorithms such as k-means. However, DAC is exact, deterministic and parameter-free in contrast to k-means.

Another limitation of DAC as a subclass of HAC is that the clustering cannot be undone. Thus, when one point is assigned to a cluster, the assignment cannot be removed or changed. This limitation does not exist in some other well-known clustering algorithms such as k-means (when the centroid of a cluster is changed, some data-point assignments will also be changed).

The HAC family of algorithms is not an attractive approach for clustering data streams because of high computational costs. Also, not all members of this family can be adapted to data streams because of their nature. DAC has this inherited limitation, although single-linkage of DAC, for instance, can be utilized for clustering data streams (limited to discrete finite space). Another most-common form of data is time-series data which have several applications. Similar to data stream applications, the costly nature of HAC argues against its use in time-series application. This issue can also be considered an inherited limitation in the proposed algorithm.

5.3 Future Work

The proposed clustering process is a general approach with remarkable extensibility. There are numerous ways to enhance this research according to those discussed in Section 5.2. Several potential directions for future work are:

- To apply the algorithm on various datasets with different types of attributes. As mentioned in Table 5.1, the proposed approach is designed for finite discrete spaces. In this thesis, the algorithm has been applied on some datasets with real and integer types of data. One possible future research direction would be to adapt this algorithm to other spaces. To this end, one may need to design a new combinatorial data structure and define a discretizing technique or apply different approaches for mapping to Hamming space.
- To construct an approach for finding optimal input parameters of the mapping (that is a patch to the proposed algorithm) data from $\mathbb{R}^D$ to $\{0,1\}^d$. This approach could increase the accuracy of clustering in $\mathbb{R}^D$ with DAC while minimizing accuracy loss.

- To incorporate other types of data inputs. In DAC, the main focus in this research was on batch clustering (clustering data that is already available). Stream data and time-series solutions have not been investigated. Although the nature of HAC is computationally expensive, the advantages of this family of algorithms make such a research direction worth investigating. For this particular purpose, the algorithm should be re-designed to make it capable of incremental clustering, which would be challenging.

References

[1] 10 key marketing trends for 2017.
https://public.dhe.ibm.com/common/ssi/ecm/wr/en/wrl12345usen/
watson-customer-engagement-watson-marketing-wr-other-papers-and-reports-wrl12345u
pdf. (Accessed on 05/23/2018).

[2] The big data boom — automobile magazine.
http://www.automobilemag.com/news/the-big-data-boom/.
(Accessed on 05/23/2018).

[3] dashee87.github.io. https://dashee87.github.io/. (Accessed on 10/12/2018).

[4] The global state of the internet in april 2017.
https://thenextweb.com/contributors/2017/04/11/
current-global-state-internet/#.tnw_iUhkTTm1. (Accessed on 05/23/2018).

[5] UCI machine learning repository.
http://archive.ics.uci.edu/ml/index.php. (Accessed on 05/25/2018).

[6] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *ArXiv preprint ArXiv:1609.08675*, 2016.

[7] Charu C Aggarwal and Chandan K Reddy. *Data clustering: algorithms and applications*. CRC press, 2013.

[8] Ahrned Najeeb Khalaf Albatineh. On similarity measures for cluster analysis. 2004.

[9] Paul K Amalaman and Christoph F Eick. Avalanche: A hierarchical, divisive clustering algorithm. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 296–310. Springer, 2015.

[10] Amineh Amini, Teh Ying Wah, and Hadi Saboohi. On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology*, 29(1):116–141, 2014.

[11] Michael R Anderberg. *Cluster analysis for applications: probability and mathematical statistics: a series of monographs and textbooks*, volume 19. Academic press, 2014.

[12] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[13] Bill Andreopoulos, Aijun An, and Xiaogang Wang. Hierarchical density-based clustering of categorical data and a simplification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 11–22. Springer, 2007.

[14] Periklis Andritsos, Panayiotis Tsaparas, Renée J Miller, and Kenneth C Sevcik. Limbo: Scalable clustering of categorical data. In *International Conference on Extending Database Technology*, pages 123–146. Springer, 2004.

[15] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod Record*, volume 28, pages 49–60. ACM, 1999.

[16] A Baccini, Ph Besse, and A Falguerolles. A l1-norm pca and a heuristic approach. *Ordinal and Symbolic Data Analysis*, 1(1):359–368, 1996.

[17] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems*, pages 585–591, 2002.

[18] Mikhail Belkin and Partha Niyogi. Using manifold stucture for partially labeled classification. In *Advances in Neural Information Processing Systems*, pages 953–960, 2003.

[19] R Bellman. Curse of dimensionality. *Adaptive Control Processes: a Guided Tour. Princeton, NJ*, 1961.

[20] Tamara L Berg and Alexander C Berg. Finding iconic images. In *Computer Vision and Pattern Recognition Workshops, 2009. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2009.

[21] James C Bezdek. Objective function clustering. In *Pattern Recognition with Fuzzy Objective Function Algorithms*, pages 43–93. Springer, 1981.

[22] Daniel Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.

[23] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[24] Magnus Borga and Tomas Landelius Hans Knutsson. A unifed approach to pca, pls, mlr and cca.

[25] Nizar Bouguila. On multivariate binary data clustering and feature weighting. *Computational Statistics & Data Analysis*, 54(1):120–134, 2010.

[26] Nizar Bouguila and Khalid Daoudi. A statistical approach for binary vectors modeling and clustering. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 184–195. Springer, 2009.

[27] Matthew Brand. Charting a manifold. In *Advances in Neural Information Processing Systems*, pages 985–992, 2003.

[28] Jörg Bruske and Gerald Sommer. Intrinsic dimensionality estimation with optimally topology preserving maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):572–575, 1998.

[29] Christopher JC Burges et al. Dimension reduction: A guided tour. *Foundations and Trends® in Machine Learning*, 2(4):275–365, 2010.

[30] Cesar F Caiafa and Andrzej Cichocki. Multidimensional compressed sensing and their applications. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 3(6):355–380, 2013.

[31] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):11, 2011.

[32] Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[33] Emmanuel J Candes and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE Transactions on Information Theory*, 52(12):5406–5425, 2006.

[34] Emmanuel J Candès and Michael B Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, 2008.

[35] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1, 2007.

[36] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of The Thiry-Fourth Annual ACM Symposium on Theory of Computing*, pages 380–388. ACM, 2002.

[37] Peter Cheeseman, James Kelly, Matthew Self, John Stutz, Will Taylor, and Don Freeman. Autoclass: A bayesian classification system. In *Machine Learning Proceedings 1988*, pages 54–64. Elsevier, 1988.

[38] Bernard Chen, Phang C Tai, Robert Harrison, and Yi Pan. Novel hybrid hierarchical-k-means clustering method (hk-means) for microarray analysis. In *Computational Systems Bioinformatics Conference, 2005. Workshops and Poster Abstracts. IEEE*, pages 105–108. IEEE, 2005.

[39] Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, and Edward Y Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):568–586, 2011.

[40] Eng Yeow Cheu, Chee Keongg, and Zonglin Zhou. On the two-level hybrid clustering algorithm. In *International Conference on Artificial Intelligence in Science and Technology*, pages 138–142, 2004.

[41] Hugh Chipman and Robert Tibshirani. Hybrid hierarchical clustering with applications to microarray data. *Biostatistics*, 7(2):286–301, 2005.

[42] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.

[43] Vartan Choulakian. L1-norm projection pursuit principal component analysis. *Computational Statistics & Data Analysis*, 50(6):1441–1451, 2006.

[44] Cheng-Tao Chu, Sang K Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Y Ng. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems*, pages 281–288, 2007.

[45] Ondrej Chum and Jiri Matas. Web scale image clustering–large scale discovery of spatially related images. 2008.

[46] Michael Collins, Sanjoy Dasgupta, and Robert E Schapire. A generalization of principal components analysis to the exponential family. In *Advances in Neural Information Processing Systems*, pages 617–624, 2002.

[47] John P Cunningham and Zoubin Ghahramani. Linear dimensionality reduction: survey, insights, and generalizations. *Journal of Machine Learning Research*, 16(1):2859–2900, 2015.

[48] Mô Dang and Gérard Govaert. Fuzzy clustering of spatial binary data. *Kybernetika*, 34(4):393–398, 1998.

[49] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of The 16th International Conference on World Wide Web*, pages 271–280. ACM, 2007.

[50] Manoranjan Dash, Huan Liu, and Xiaowei Xu. Merging distance and density based clustering. In *Database Systems for Advanced Applications, 2001. Proceedings. Seventh International Conference on*, pages 32–39. IEEE, 2001.

[51] Manoranjan Dash, Simona Petrutiu, and Peter Scheuermann. ppop: Fast yet accurate parallel hierarchical clustering using partitioning. *Data & Knowledge Engineering*, 61(3):563–578, 2007.

[52] William HE Day and Herbert Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1(1):7–24, 1984.

[53] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[54] Daniel Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[55] Pierre Demartines and Jeanny Hérault. Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets. *IEEE Transactions on Neural Networks*, 8(1):148–154, 1997.

[56] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of The Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[57] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *The Conference on Computer Vision and Pattern Recognition*, 2009.

[58] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 551–556. ACM, 2004.

[59] Inderjit S Dhillon and Dharmendra S Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large-Scale Parallel Data Mining*, pages 245–260. Springer, 2002.

[60] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, volume 32, pages 647–655, 2014.

[61] David L Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[62] David L Donoho and Carrie Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

[63] Matthijs Douze, Hervé Jégou, Harsimrat Sandhawalia, Laurent Amsaleg, and Cordelia Schmid. Evaluation of gist descriptors for web-scale image search. In *Proceedings of The ACM International Conference on Image and Video Retrieval*, page 19. ACM, 2009.

[64] Joseph C Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. 1973.

[65] Carl Eckart and Gale Young. A principal axis transformation for non-hermitian matrices. *Bulletin of the American Mathematical Society*, 45(2):118–121, 1939.

[66] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Knowledge Discovery and Data Mining*, volume 96, pages 226–231, 1996.

[67] Brian S Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. Hierarchical clustering. *Cluster Analysis, 5th Edition*, pages 71–110, 2011.

[68] Rob Fergus, Yair Weiss, and Antonio Torralba. Semi-supervised learning in gigantic image collections. In *Advances in Neural Information Processing Systems*, pages 522–530, 2009.

[69] Douglas H Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[70] Imola K Fodor. A survey of dimension reduction techniques. Technical report, Lawrence Livermore National Lab., CA (US), 2002.

[71] George Forman and Bin Zhang. Distributed data clustering can be efficient and exact. *ACM SIGKDD Explorations Newsletter*, 2(2):34–38, 2000.

[72] Jan-Michael Frahm, Pierre Fite-Georgel, David Gallup, Tim Johnson, Rahul Raguram, Changchang Wu, Yi-Hung Jen, Enrique Dunn, Brian Clipp, Svetlana Lazebnik, et al. Building rome on a cloudless day. In *European Conference on Computer Vision*, pages 368–381. Springer, 2010.

[73] Olivier François, Sophie Ancelet, and Gilles Guillot. Bayesian clustering using hidden markov random fields in spatial population genetics. *Genetics*, 174(2):805–816, 2006.

[74] Keinosuke Fukunaga and David R Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, 100(2):176–183, 1971.

[75] Jacqueline S Galpin and Douglas M Hawkins. Methods of l1 estimation of a covariance matrix. *Computational Statistics & Data Analysis*, 5(4):305–319, 1987.

[76] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):744–755, 2014.

[77] Dušan Gleich. Learning based data mining using compressed sensing. In *Geoscience and Remote Sensing Symposium (IGARSS), 2014 IEEE International*, pages 1643–1646. IEEE, 2014.

[78] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, 2013.

[79] Yunchao Gong, Marcin Pawlowski, Fei Yang, Louis Brandy, Lubomir Bourdev, and Rob Fergus. Web scale photo hash clustering on a single machine. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 19–27, 2015.

[80] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, Reading, Massachusetts, 1994.

[81] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *ACM Sigmod Record*, volume 27, pages 73–84. ACM, 1998.

[82] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.

[83] RB Hakimi, E Winarko, et al. Clustering binary data based on rough set lndiscernibility level. *Interational Joural of Biomedical Soil Computing and Human Sciences*, 16(2), 2010.

[84] Timothy J Harlow, J Peter Gogarten, and Mark A Ragan. A hybrid clustering approach to recognition of protein families in 114 microbial genomes. *BMC Bioinformatics*, 5(1):45, 2004.

[85] Kaiming He, Fang Wen, and Jian Sun. K-means hashing: An affinity-preserving quantization method for learning binary compact codes. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, pages 2938–2945, 2013.

[86] Zengyou He, Xiaofei Xu, and Shengchun Deng. Squeezer: an efficient algorithm for clustering categorical data. *Journal of Computer Science and Technology*, 17(5):611–624, 2002.

[87] Christian Hennig, Marina Meila, Fionn Murtagh, and Roberto Rocci. *Handbook of cluster analysis.* CRC Press, 2015.

[88] Alexander Hinneburg and Daniel A Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. 1999.

[89] Alexander Hinneburg, Daniel A Keim, et al. An efficient approach to clustering in large multimedia databases with noise. In *Knowledge Discovery and Data Mining*, volume 98, pages 58–65, 1998.

[90] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3):283–304, 1998.

[91] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[92] Aapo Hyvarinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999.

[93] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of The Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613. ACM, 1998.

[94] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[95] Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. Dbdc: Density based distributed clustering. In *International Conference on Extending Database Technology*, pages 88–105. Springer, 2004.

[96] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[97] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: re-rank with source coding. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 861–864. IEEE, 2011.

[98] Michel Journée, Yurii Nesterov, Peter Richtárik, and Rodolphe Sepulchre. Generalized power method for sparse principal component analysis. *Journal of Machine Learning Research*, 11(Feb):517–553, 2010.

[99] U Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *Proceedings of The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 316–324. ACM, 2012.

[100] U Kang, Charalampos E Tsourakakis, and Christos Faloutsos. Pegasus: mining peta-scale graphs. *Knowledge and Information Systems*, 27(2):303–325, 2011.

[101] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515, 2004.

[102] Kari Karhunen and Ivan Selin. On linear methods in probability theory. 1960.

[103] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[104] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[105] Gunhee Kim, Christos Faloutsos, and Martial Hebert. Unsupervised modeling of object categories using link analysis techniques. In *Computer Vision and Pattern Recognition, 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[106] Benjamin King. Step-wise clustering procedures. *Journal of the American Statistical Association*, 62(317):86–101, 1967.

[107] Donald Knuth. *The TEXbook.* Addison-Wesley, Reading, Massachusetts, 1986.

[108] Teuvo Kohonen. The self-organizing map. *Neurocomputing*, 21(1-3):1–6, 1998.

[109] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[110] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[111] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[112] Brian Kulis et al. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364, 2013.

[113] D Ashok Kumar and Loraine Charlet Annie. Binary data clustering based on wiener transformation. In *Pattern Recognition, Informatics and Medical Engineering (PRIME)*, pages 55–60. IEEE, 2012.

[114] Charlotte Laclau and Mohamed Nadif. Fast simultaneous clustering and feature selection for binary data. In *International Symposium on Intelligent Data Analysis*, pages 192–202. Springer, 2014.

[115] Leslie Lamport. *LATEX — A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[116] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of The IEEE*, 86(11):2278–2324, 1998.

[117] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[118] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction.* Springer Science & Business Media, 2007.

[119] Yong Jae Lee and Kristen Grauman. Shape discovery from unlabeled image collections. In *Computer Vision and Pattern Recognition, 2009. IEEE Conference on*, pages 2254–2261. IEEE, 2009.

[120] Elizaveta Levina and Peter J Bickel. Maximum likelihood estimation of intrinsic dimension. In *Advances in Neural Information Processing Systems*, pages 777–784, 2005.

[121] Tao Li. A general model for clustering binary data. In *Proceedings of The Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 188–197. ACM, 2005.

[122] Tao Li. A unified view on clustering binary data. *Machine Learning*, 62(3):199–215, 2006.

[123] Xiaobo Li. Parallel algorithms for hierarchical clustering and cluster validity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(11):1088–1092, 1990.

[124] Xiaobo Li and Zhixi Fang. Parallel clustering algorithms. *Parallel Computing*, 11(3):275–290, 1989.

[125] Chuan Liu, Wenyong Wang, Martin Konan, Siyang Wang, Lisheng Huang, Yong Tang, and Xiang Zhang. A new validity index of feature subset for evaluating the dimensionality reduction algorithms. *Knowledge-Based Systems*, 121:83–98, 2017.

[126] Ting Liu, Charles Rosenberg, and Henry A Rowley. Clustering billions of images with large scale nearest neighbor search. In *Applications of Computer Vision, 2007. WACV'07. IEEE Workshop on*, pages 28–28. IEEE, 2007.

[127] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *Computer Vision and Pattern Recognition, 2012 IEEE Conference on*, pages 2074–2081. IEEE, 2012.

[128] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *Proceedings of The 28th International Conference on Machine Learning (ICML-11)*, pages 1–8, 2011.

[129] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 911–916. IEEE, 2010.

[130] Stuart Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[131] Roberto J López-Sastre, Daniel Oñoro-Rubio, Pedro Gil-Jiménez, and Saturnino Maldonado-Bascón. Fast reciprocal nearest neighbors clustering. *Signal Processing*, 92(1):270–275, 2012.

[132] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[133] Louis L McQuitty. Elementary linkage analysis for isolating orthogonal and oblique types and typal relevancies. *Educational and Psychological Measurement*, 17(2):207–229, 1957.

[134] Marina Meilă. Comparing clusteringsan information based distance. *Journal of Multivariate Analysis*, 98(5):873–895, 2007.

[135] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In *Neural Networks for Signal Processing IX, 1999. Proceedings of The 1999 IEEE Signal Processing Society Workshop.*, pages 41–48. Ieee, 1999.

[136] Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. *The Quarterly Journal of Mathematics*, 11(1):50–59, 1960.

[137] Shakir Mohamed, Zoubin Ghahramani, and Katherine A Heller. Bayesian exponential family pca. In *Advances in Neural Information Processing Systems*, pages 1089–1096, 2009.

[138] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[139] Harsha S Nagesh, Sanjay Goil, and Alok Choudhary. A scalable parallel subspace clustering algorithm for massive data sets. In *Parallel Processing, 2000. Proceedings. 2000 International Conference on*, pages 477–484. IEEE, 2000.

[140] Satyasai Jagannath Nanda and Ganapati Panda. A survey on nature inspired metaheuristic algorithms for partitional clustering. *Swarm and Evolutionary Computation*, 16:1–18, 2014.

[141] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2002.

[142] Raymond T. Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[143] Mohammad Norouzi and David M Blei. Minimal loss hashing for compact binary codes. In *Proceedings of The 28th International Conference on Machine Learning (ICML-11)*, pages 353–360, 2011.

[144] Mohammad Norouzi and David J Fleet. Cartesian k-means. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, pages 3017–3024, 2013.

[145] Mohammad Norouzi, Ali Punjani, and David J Fleet. Fast search in hamming space with multi-index hashing. In *Computer Vision and Pattern Recognition, 2012 IEEE Conference on*, pages 3108–3115. IEEE, 2012.

[146] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[147] Clark F Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 21(8):1313–1325, 1995.

[148] Edward Omiecinski and Peter Scheuermann. A parallel algorithm for record clustering. *ACM Transactions on Database Systems (TODS)*, 15(4):599–624, 1990.

[149] Carlos Ordonez. Clustering binary data streams with k-means. In *Proceedings of The 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 12–19. ACM, 2003.

[150] Francesco Pappalardo, Cristiano Calonaci, Marzio Pennisi, Emilio Mastriani, and Santo Motta. Hamfast: fast hamming distance computation. In *Computer Science and Information Engineering, 2009 World Congress on*, volume 1, pages 569–572. IEEE, 2009.

[151] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[152] Karl W Pettis, Thomas A Bailey, Anil K Jain, and Richard C Dubes. An intrinsic dimensionality estimator from near-neighbor information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (1):25–37, 1979.

[153] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Computer Vision and Pattern Recognition, 2007. IEEE Conference on*, pages 1–8. IEEE, 2007.

[154] Rahul Raguram and Svetlana Lazebnik. Computing iconic summaries of general visual concepts. In *Computer Vision and Pattern Recognition Workshops, 2008. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2008.

[155] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[156] F James Rohlf. Single-link clustering algorithms. *Handbook of Statistics*, 2:267–284, 1982.

[157] Lior Rokach. A survey of clustering algorithms. In *Data mining and knowledge discovery handbook*, pages 269–298. Springer, 2009.

[158] Sam T Roweis. Em algorithms for pca and spca. In *Advances in Neural Information Processing Systems*, pages 626–632, 1998.

[159] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[160] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *RBM*, 500(3):500, 2007.

[161] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[162] John W Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 100(5):401–409, 1969.

[163] Tapesh Santra. A bayesian non-parametric method for clustering high-dimensional binary data. *ArXiv preprint ArXiv:1603.02494*, 2016.

[164] Shriram Sarvotham, Dror Baron, and Richard G Baraniuk. Measurements vs. bits: Compressed sensing meets information theory. In *Allerton Conference on Communication, Control and Computing*, 2006.

[165] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[166] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[167] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813, 2014.

[168] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB*, volume 98, pages 428–439, 1998.

[169] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[170] Robin Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[171] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep fisher networks for large-scale image classification. In *Advances in Neural Information Processing Systems*, pages 163–171, 2013.

[172] Marek Śmieja, Krzysztof Hajto, and Jacek Tabor. Efficient mixture model for clustering of sparse high dimensional binary data. *ArXiv preprint ArXiv:1707.03157*, 2017.

[173] Peter HA Sneath, Robert R Sokal, et al. *Numerical taxonomy. The principles and practice of numerical classification.* 1973.

[174] John Snow. *On the mode of communication of cholera.* John Churchill, 1855.

[175] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*, 2014.

[176] Michael Steinbach, George Karypis, Vipin Kumar, et al. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, volume 400, pages 525–526. Boston, 2000.

[177] Darius Tamasauskas, Virgilijus Sakalauskas, and Dalia Kriksciuniene. Evaluation framework of hierarchical clustering methods for binary data. In *Hybrid Intelligent Systems (HIS), 2012 12th International Conference on*, pages 421–426. IEEE, 2012.

[178] Yang Tang, Ryan P Browne, and Paul D McNicholas. Model based clustering of high-dimensional binary data. *Computational Statistics & Data Analysis*, 87:84–101, 2015.

[179] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[180] Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

[181] Mark J Van der Laan and Katherine S Pollard. A new algorithm for hybrid hierarchical clustering with visualization and the bootstrap. *Journal of Statistical Planning and Inference*, 117(2):275–303, 2003.

[182] Peter J. Verveer and Robert P. W. Duin. An evaluation of intrinsic dimensionality estimators. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):81–86, 1995.

[183] Michail Vlachos, Carlotta Domeniconi, Dimitrios Gunopulos, George Kollios, and Nick Koudas. Non-linear dimensionality reduction techniques for classification and visualization. In *Proceedings of The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 645–651. ACM, 2002.

[184] Michail Vlachos, Nikolaos M Freris, and Anastasios Kyrillidis. Compressive mining: fast and optimal data mining in the compressed domain. *The VLDB Journal*, 24(1):1–24, 2015.

[185] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[186] Jianzhong Wang. *Geometric structure of high-dimensional data and dimensionality reduction*. Springer, 2011.

[187] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *Computer Vision and Pattern Recognition, 2010 IEEE Conference on*, pages 3424–3431. IEEE, 2010.

[188] Li Wang and Zheng-Ou Wang. Cubn: A clustering algorithm based on density and distance. In *Machine Learning and Cybernetics, 2003 International Conference on*, volume 1, pages 108–112. IEEE, 2003.

[189] Wei Wang, Jiong Yang, Richard Muntz, et al. Sting: A statistical information grid approach to spatial data mining. In *VLDB*, volume 97, pages 186–195, 1997.

[190] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of The American Statistical Association*, 58(301):236–244, 1963.

[191] Matthijs Joost Warrens et al. *Similarity coefficients for binary data: properties of coefficients, coefficient matrices, multi-way metrics and multivariate coefficients*. Psychometrics and Research Methodology Group, Leiden University Institute for Psychological Research, Faculty of Social Sciences, Leiden University, 2008.

[192] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, pages 1753–1760, 2009.

[193] Max Welling, Christopher Williams, and Felix V Agakov. Extreme components analysis. In *Advances in Neural Information Processing Systems*, pages 137–144, 2004.

[194] Christopher KI Williams and Felix V Agakov. Products of gaussians and probabilistic minor component analysis. *Neural Computation*, 14(5):1169–1182, 2002.

[195] William Thomas Williams and Jean M Lambert. Multivariate methods in plant ecology: I. association-analysis in plant communities. *The Journal of Ecology*, pages 83–101, 1959.

[196] Stephen Winters-Hilt and Sam Merat. Svm clustering. In *BMC bioinformatics*, volume 8, page S18. BioMed Central, 2007.

[197] David Wishart. 256. note: An algorithm for hierarchical classifications. *Biometrics*, pages 165–170, 1969.

[198] M Anthony Wong. A hybrid clustering method for identifying high-density clusters. *Journal of the American Statistical Association*, 77(380):841–847, 1982.

[199] Dongbin Xiu. *Numerical methods for stochastic computations: a spectral method approach.* Princeton university press, 2010.

[200] Xiaowei Xu, Martin Ester, H-P Kriegel, and Jörg Sander. A distribution-based clustering algorithm for mining in large spatial databases. In *Data Engineering, 1998. Proceedings., 14th International Conference on*, pages 324–331. IEEE, 1998.

[201] Yiling Yang, Xudong Guan, and Jinyuan You. Clope: a fast and effective clustering algorithm for transactional data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 682–687. ACM, 2002.

[202] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[203] Hongyuan Zha, Xiaofeng He, Chris Ding, Ming Gu, and Horst D Simon. Spectral relaxation for k-means clustering. In *Advances in Neural Information Processing Systems*, pages 1057–1064, 2002.

[204] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *ACM Sigmod Record*, volume 25, pages 103–114. ACM, 1996.

[205] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on mapreduce. In *IEEE International Conference on Cloud Computing*, pages 674–679. Springer, 2009.

[206] Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286, 2006.

APPENDICES

Appendix A

Dimensionality Reduction

Dimensionality Reduction and Feature Compression

Assume that a penny dropped somewhere on a straight line that is 100 yards long. The penny would not be too hard to find. Just by walking along the line. It would take two minutes.

Now say there is a square 100 yards on each side. Finding a penny dropped somewhere on it would be very hard, like searching across two football fields stuck together. It could take days.

Now consider searching a cube 100 yards across. That would be like searching a 30-story building the size of a football stadium.

The difficulty of searching through space gets a *lot* harder as more dimensions are added. The fact may not be intuitively obvious when it is just stated in mathematical formulas since they all have the same width. That is the "curse of dimensionality" named so because the term is unintuitive, useful, and yet simple.

Imagine the data that the world uses each day. Images, sounds, documents, videos are just a few examples of this data. These data types are only a hint of a more significant challenge. All these forms of data need to be analyzed and processed in order to be managed. These data types come in different dimensions, but high-dimensionality is common in all of them. Processing or analyzing high-dimensional data is not easy. Therefore, some techniques have been proposed to reduce the dimensionality of data. These dimensionality reduction techniques play a crucial role in almost all data science fields. Representing high-dimensional data by its low-dimensional version is the goal of these methods because the low-dimensional version of data can be used for among other things, better understanding (normally in feature selection techniques) or helping resource usage in analyzing and processing the data.

Here, the main benefits of dimensionality reduction and feature compression are listed:

- Obviously, storing high-dimensional data needs more resources. Therefore, dimensionality reduction, which is a form of data compression, can effectively help reduce storage costs.

- It is not only storage costs that increase with higher dimensional data, but also the resources necessary for processing this data. One simple example is that when processing memory cannot handle the large size data, the processor needs to go back and forth to external storages, which affects both time and system resources. Thus, having lower-dimensional data can help reduce resource usage cost.

- Besides, the example about finding a coin (model) in a cubic 100-yard space, which also shows the drawback of higher dimensions with sparse feature space, one can think of a non-sparse feature space example. Assume that there is a feature space where one feature is the length of an iron bar. Other features are the temperature of the environment in Celsius (the second feature), in Fahrenheit (the third feature), and in Kelvin (the fourth feature). Having a highly correlated, or redundant features does not help pattern recognition, which in this example is a simple linear function between temperature and length of the bar no matter which temperature unit is selected. If there exist sufficient resources to derive a model for this overly-complicated feature space, interpreting the result of this numerically unstable model becomes difficult.

- Consider the iron bar example again, but with only three features. The first two are the same: the bar's length and the test bench temperature in Celsius. The third feature is the temperature outside the lab, which is constant during the experiment. Therefore, the feature space is three-dimensional. If one wants to visualize this space, the result is hard to interpret in three dimensions, but by removing the third feature data visualization of the two-dimensional space becomes easy. Also, having an uncorrelated feature does not help pattern recognition between features.

- Some methods for dealing with noise and outliers are done in a pre-processing step before the dimensionality reduction stage. However, some other techniques can merge dimensionality reduction with noise and outliers' removal. In these techniques, the noise handling is more robust, which makes dimensionality reduction a useful tool for dealing with noise removal tasks.

All being said, dimensionality reduction helps all data mining tasks, visualization, and compression of high-dimensional data. As dimensionality reduction can be considered an old problem, several approaches have been proposed for it: from traditional linear approaches that can handle simple close-to-linear data to nonlinear dimensionality reduction techniques that can handle complex non-linear data. The problem and proposed solutions are so vast that several books have been published about them. Therefore, digging into the details of all techniques is beyond the scope of this thesis, but the concept behind some techniques are described to clarify the proposed approach (i.e., DAC).

Dimensionality Reduction

Dimensionality Reduction (DR) plays an important role in data science. In order to obtain the required accuracy, the sample size grows exponentially by the number of variables (features or dimensions), which is termed the curse of dimensionality by Bellman [19]. Fortunately, there are cures for this curse, some of which are named dimensionality reduction.

Commonness of High Dimensional Data

In the modern Information world, many objects have digital representation. Images, speech, videos, documents, handwritten texts and signatures, and even human emotions have their electronically stored versions. Not only is storing these enormous amounts of data but processing them is also crucial to extract knowledge and sometimes come to a decision from them. Extracting knowledge could be done by a natural language processing feature on an automatic answering machine, and decision making could involve opening a security door through fingerprint scanning. In many cases, either local computational resources are limited, or the computational costs are not feasible for using powerful machines to process large data. Although both colossal numbers of data instances and the dimension of each instance can result in drastically huge data, the latter has worse impacts. Therefore, reducing the dimensions of the data without losing much information becomes crucial. Here, some examples of high dimensional data are brought to show the commonness of this issue.

Lane keep assistant systems are used in many recent cars. Assume that this system uses 256×256 RGB images. Each side of a car has a camera that captures an image, say, every 0.5 seconds (which is far less than real-world cars do). Thus, the lane keep assistant alone takes four pictures every second each three 65,536 dimensional vectors. If dimensionality reduction is not applied to this enormous amount of data, processing would need a very sturdy system. **Collision prediction**, **pedestrian action recognition**, and **cross traffic assist** systems need at least two wide cameras to monitor activity in the road ahead and provide preventative assistance to the driver when it is needed most.

Document search or machine learning on documents is based on the content of a document, that is, the words which are its meaningful elements. If a document includes n words, it can be represented as a vector with n dimensions. If each page of a document includes 200 words, on average, a short version of a paper is represented by a 1200-dimension vector and a 200-page book is represented by a vector with 40,000 dimensions. Thus, for large documents, dimensionality reduction is an essential task for any search or machine learning algorithm.

Face recognition is currently used in some applications or even some smartphones for security reasons. Assume that the resolution of a face picture is 256×256. The system should process a vector with 65,536 dimensions. Storing and processing just a single picture on a smartphone is feasible, but in many systems, this dimension is a threat to the processing time. Therefore, in this case, also dimensionality reduction is inevitable. These are just a few examples of the commonness of high-dimensional data and the reason behind dimensionality reduction. There follows the definition of the dimension of data and the types of dimensions that exist.

Given a set of records $X = \{\mathbf{x}_i\}_{i=1}^{n}$ in an original space S_o with D dimension ($\mathbb{R}^D$), assume that one can have a d dimensional manifold M_t that is embedded in S_o. The **extrinsic** dimension of data would be D and the **intrinsic** dimension of data is d. Therefore, if there exists such a manifold, the dimensionality reduction of data is achievable. Thus, one can define the dimensionality reduction of data by deriving manifold parameters for finding the intrinsic dimension of each data point.

Finding the intrinsic dimension of a dataset is important in dimensionality reduction, but this estimation is not straightforward as multiple manifolds may be embedded in the original space, and those manifolds can obviously have different dimensions. Simplest manifolds are hyperplanes or linear manifolds. One major positive characteristic of this type of manifold is that the similarity between data points in the original space is preserved after the data is transferred onto the manifold. Besides, transferring back from a target space (i.e., manifold) to the original space is straightforward, which can be important in some applications. If the dataset has a linear geometric structure, using linear intrinsic dimensions can also be helpful for eliminating outliers in data. However, if the structure of data is non-linear, it is evident that using linear manifolds is useless, and consequently, other ideas such as manifold projection techniques [159, 179, 17, 62, 27, 152] have been proposed. The dimension of the embedding is an indispensable factor for manifold projection techniques, as too-small dimensions would collapse features onto the same dimension and too-large dimensions would make the projection noisy [120]. Various methods use various approaches; for example, *Locally-Linear Embedding (LLE)* [159] assumes that the user provides the intrinsic dimension. All being said, estimating the topological dimension of the data is crucial, yet not easy for a finite set of data.

One can categorize intrinsic dimension estimation methods into two groups: eigenvalue or projection methods, and geometric approaches [186]. Projection techniques [74, 28, 182] use the neighbourhood structure of data to estimate its intrinsic dimension. These methods are also widely used in data mining approaches to increase the efficiency of data mining algorithms [18, 183], although these approaches have the drawback of results sensitivity to neighbourhood size. The geometric approaches are based on the fractal dimensions of the nearest neighbour distances [186]. In contrast to projection methods that are often used for exploratory reasons and sometimes provide unreliable dimension estimations, geometric approaches reliably estimate intrinsic dimensions. A detailed discussion of the estimation of intrinsic dimensions is beyond the scope of this thesis.

There are several classifications of dimensionality reduction techniques. One classification is based on the extrinsic dimension of the data. Many pattern recognition and machine learning approaches use data points that have hundreds of thousands of dimensions, and no dimension has an explicit interpretation. This enormous dimension of data needs to be reduced to smaller numbers so that data mining becomes feasible. This size of dimensionality reduction is called the *hard* dimensionality reduction. In contrast, there are applications, basically in statistical analysis, in which the extrinsic (i.e. original space) dimensions are drastically fewer (i.e., a few tens) and the value of each dimension has a meaning and interpretation, so the reduction required is not too much. This class is called the *soft* dimensionality reduction group of techniques.

One can classify DR techniques by different criteria, but the most well-known classification of DR approaches is by linearity. This venue is followed in the current research and these techniques are grouped into *linear dimensionality reduction* vs *non-linear dimensionality reduction*. However, before that, it is worth mentioning the Kosambi-Karhunen-Loeve theorem and the Johnson-Lindenstrauss lemma that are the foundations of dimensionality reduction techniques.

Kosambi-Karhunen-Loeve Theorem

Like Fourier series representation of a function on a bounded interval, Kosambi-Karhunen-Loeve is a representation of a stochastic process as an infinite linear combination of orthogonal functions. This transformation [102] which is closely related to principal component analysis, also called eigenvector or Hotelling transform. It is widely used in numerous data science applications. Many of such expansions exist for a stochastic process, but the Karhunen-Loeve gives the minimum mean square error which makes this transformation important. This expansion is one of the most widely used techniques in dimensionality reduction. However, the mathematical description of Karhunen-Loeve expansion has preliminaries on compact operators, Mercer's theorem, and stochastic processes which are beyond the framework of this thesis. The Karhunen-Loeve expansion is not going to be discussed further here. Interested reader may refer to [199].

Johnson-Lindenstrauss Lemma

Given $0 < \varepsilon < 1$ $0 < \varepsilon < 1$, a set X of m points in $\mathbb{R}^N$, and a number $n > 8\ln(m)/\varepsilon^2$, there is a linear map $f : \mathbb{R}^N \to \mathbb{R}^n$ such that

$$(1-\varepsilon)\|u-v\|^2 \leq \|f(u)-f(v)\|^2 \leq (1+\varepsilon)\|u-v\|^2 \tag{A.1}$$

for all $u, v \in X$.

The formula can be rearranged:

$$(1+\varepsilon)^{-1}\|f(u)-f(v)\|^2 \leq \|u-v\|^2 \leq (1-\varepsilon)^{-1}\|f(u)-f(v)\|^2 \tag{A.2}$$

The Johnson-Lindenstrauss lemma is named after William B. Johnson and Joram Lindenstrauss concerning low-distortion embeddings of points from high-dimensional into low-dimensional Euclidean space. The lemma states that a small set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved. This lemma is used in dimensionality reduction as well as manifold learning, compressed sensing, and graph embedding which are helpful for the nearest-neighbour search, information retrieval, and many data mining application. The proof of JohnsonLindenstrauss lemma is outside the scope of this chapter.

Dimensionality Reduction VS Data Compression

As mentioned in previous sections, dimensionality reduction is inevitable and imperative in many data-related fields. It is also mentioned that many features are duplicated, heavily correlated with other features, or small information-added features. Therefore, one idea is to select informative features within data, which is called *feature selection*. Under another idea, *feature extraction*, users do not select features but transfer them to another coordinates or transform them into other sets of coordinates with lower dimensionality to give better structure, which also helps in data interpretation. All these tasks are done before performing any other statistical or data mining processes; therefore, together they are sometimes referred to a *preprocessing*. Another approach is to reduce the population numbers (which are bits in many domains) needed to represent the data. This approach is called *data compression*.

Many dimensionality reduction techniques can be considered an unsupervised learning method that extracts and interprets the structure of data. Like unsupervised learning methods, which have an exploratory goal, dimensionality reduction also requires investigation, but in high dimensional spaces that are possibly sparsely populated. In some statistical or machine learning approaches or many data storing and transferring situations, it is also advantageous to compress data without losing much information. Like dimensionality reduction, several approaches have been proposed for data compression. The idea of data compression came from the information theory field but currently has other applications, especially in the data mining domain. It is worth mentioning that the result of dimensionality reduction techniques can be used as the input for data compression techniques. Also, some approaches lie in the intersection of both dimensionality reduction and data compression techniques. A more recent approach was proposed in [32], whereby the data acquisition is performed in a compression scheme; therefore, there is no need for data compression after data acquisition. This approach, called *compressive sensing* (also compressed sensing, compressive sampling, or sparse sampling) also has some application in the data mining field [77, 184, 30] and has motivated research activities. Compressive sensing is outside the scope of this thesis, and interested readers may refer to [32, 33, 34, 61, 164]. In the next sections, the dimensionality reduction and data compression techniques are further discussed.

Techniques for Dimensionality Reduction

Several dimensionality reduction techniques have been proposed in the literature. Obviously, each one can have its own advantages and disadvantages. Therefore, approaches are needed in order to evaluate each DR technique. Generally, dimensionality reduction techniques are validated by reconstruction error computation or cross validation techniques, although there exist approaches that do not follow a cross validation venue and use unsupervised internal validation type techniques [125]. Conventional approaches are similar to what is done in supervised learning algorithms. For example, in a classification problem, it is a conventional approach to select a portion of the data for data training and the remaining part for testing the validity of the algorithm. One can utilize naturally occurring or artificially constructed (S-curve or Swiss roll, etc.) datasets for dimensionality reduction technique evaluation.

High-dimensional input data can be in two forms. One is the high-dimensional vectors each represents a data point; the other type is in the form of similarities/dissimilarities between data points. In this thesis, clustering (and obviously DR processing) is performed on the first type of data. Assuming that the high dimensional data is given in the form of a discrete set of vectors in Euclidean space which is embedded on a manifold that finding it, is the strict goal of dimensionality reduction. There is no access to the manifold; therefore, one tries to guess it by neighbourhood information of the data. One primary tool for this manner is to use the distance between data or in a more general form similarity/dissimilarity. Therefore, by having the similarity/dissimilarity between data, one can have a sense of neighbourhood information and hopefully a discrete version of the manifold. Furthermore, the result of dimensionality reduction on this type of data should preserve both the cardinality of the dataset and similarities within input data points. It is clear that in this data type the similarities between data points should be learned to conform to the data mining task. Two significant constraints on output data (output of DR) worth just mentioning here are centralization constraint and orthogonality constraint which are needed to be satisfied in some application. Input data types or constraints are not going to be discussed further here.

The most famous classification of DR techniques is based on the linearity of the technique where DR methods have two branches: linear DR and nonlinear DR methods. The same venue is followed here and both methods are briefly discussed here. Several other grouping exist (such as deterministic and non-deterministic, or supervised and unsupervised) for DR techniques and also a mathematical basis for every one of them, but extensive discussion about these topics is outside the scope of the thesis. Many spectral dimensionality reduction approaches can be narrowed down to finding their kernels which is an interpretation of a matrix whose decomposition provides dimensionality reduction.

From the kernel point of view, in linear DR methods, the output data is the projection of the original space. Therefore, the kernel in these methods is the matrix representation of the projection (projection operator). As an example, the kernel of the PCA method is the covariance matrix of input data, and the kernel of classical multidimensional scaling is the centring Gram matrix. In nonlinear methods, the output data is the manifold coordinate representation of the original data. The kernel in these approaches is based on a data graph which can be divided into dissimilarity, and similarity weights methods. Those trying to preserve the local distance (dissimilarity) or the weights of a point with respect to its neighbours, respectively. Therefore, what makes a difference in these two approaches is that in dissimilarity approaches the kernel is constructed from a distance graph but in similarity weight methods kernel is constructed from a similarity weight graph. As mentioned before, dimensionality reduction qualification is not an easy task unless there exists a prior knowledge about the problem. In the next two sections, dimensionality reduction techniques are discussed by classifying them into linear and nonlinear approaches.

Linear Dimensionality Reduction Algorithms

One keystone in data mining and data analysis is dimensionality reduction. Within dozens of approaches to DR, linear dimensionality reduction approaches are perhaps the most used ones. They are also the most interested ones because of their simple interpretation and computation. These techniques are widely used in high-dimensional and noisy data. As mentioned, linear dimensionality reduction performs its DR by a linear mapping of high-dimensional data to lower dimensions and preserves geometry of the data in its new linear manifold. Therefore, it is a helpful tool for visualizing, cleaning, compressing of the data. Different domains and optimizations resulted in various forms of linear dimensionality reduction techniques.

One may assume that linear methods are all a generalized form of the eigenvalue problem which is not correct. Here, some approaches in linear dimensionality reduction are briefly described and the detailed discussions are left to the reader by referring to [29, 24, 47, 70]. Next is a short description of *Principle Component Analysis (PCA)*, *Multi-Dimensional Scaling (MDS)* and *Random Projections*.

The most important and widely used linear dimensionality reduction method is PCA. The global neighbourhood system, which is also the case in the PCA, is mostly used in linear DR methods. Pearson invented PCA at 1901 [151]. Various interpretations and definitions are proposed based on different point of views such as statistical or geometrical but, the approach of PCA is basically the same in all of them. PCA is the method to find the d-dimensional projection (d principal directions) of data that maximizes data (centred) variance. The DR data are obtained from *Singular Value Decomposition (SVD)* of the data matrix or *Spectral (eigenvalue) Decomposition* of the covariance matrix of data. It is worth mentioning that in cases where the dimension of data is much lower than the number of data points, eigenvalue decomposition is quite fast because of the lower dimensionality of the covariance matrix. One can say simply that PCA algorithms consist of two major parts: data centralization and applying SVD on centralized data Eckart and young [65], Mirsky [136] Several noteworthy versions of PCA have been proposed. Kernel

PCA [135] uses PCA on feature space. Probabilistic PCA Roweis [158], Tipping [180], extreme component analysis [193] are some probabilistic extensions of PCA. There are also some extensions of PCA that do not follow the linearity definition of PCA and result in the non-linear mapping of data. Interested reader can refer to Collins et al 2002 [46], Zoe et al 2006 [206], Mohamed et al 2008 [137], Journee et al 2010 [98], Candes et al 2011 [31] , Williams and Agakov 2002 [194] , Choulakian 2006 [43] , Galpin and Hawkins 1987 [75] , Hyvarinen et al 2001 [92], Baccini et al 1996 [16] , wang 2011 [186], Cunningham and Ghahramani 2015 [47], Fodor 2002 [70], Sorzano and Montano 2014 [175] for more detailed discussions on PCA extensions. It is worth noting again that PCA and some of its extensions (linear ones) are linear projection approaches which perform well when the data reside on subspace or hyperplane and perform quite poorly when the data is situated on non-linear manifolds.

If instead of minimizing the reconstruction error; that is the objective in PCA; the scatter of projection is maximized, one would get Classical Multi-Dimensional Scaling (CMDS) or simply MDS. This family of methods has a very close connection with PCA even in some special cases; classical multi-dimensional scaling and maximal variance PCA; are identical. For more information on MDS refer to [23].

Non-Linear Dimensionality Reduction Algorithms

Many conventional dimensionality reduction techniques such as classical PCA or MDS are based on linear models that are limited to specific problems (on linear manifolds). Recently, some methods have been proposed for non-linear dimensionality reduction problems (or manifold learning). Non-linear dimensionality reduction methods became a hot topic as they are more powerful than linear ones. However, non-linear techniques require more parameters than linear ones (that are basically a simple matrix multiplication). More required parameters need more input data to help computing parameters. This large amount of required data is the major drawback of *Non-Linear Dimensionality Reduction (NLDR)* techniques. Some generalized PCA [46, 137] or MDS approaches [179] can be categorized as NLDR. The topic of NLDR (manifold learning) is an interesting venue of research, but fall outside the scope of this thesis. NLDR methods can be categorized by their preservation criterion [118]: (i) *Distance PReservation (DPR)* approaches, and (ii) *Topology PReservation (TPR)* approaches.

In DPR, NLDR methods reduce the dimensionality by preserving pairwise distances as the criterion. This group of methods ensures the consistency of the global shape or the local neighbourhood relationships between original space (high-dimensional) and target space (low-dimensional). As preserving distances in non-linear transformation is not perfectly achievable, various optimization procedures result in various approaches. Two main subcategories have been mentioned for DPR: *first*, spatial distance preservation approaches such as Sammon's Mapping [162], Curvilinear Component Analysis [55], and *second*, graph distance preservation approaches such as ISOMAP [179]. Kernel PCA can also be considered in the DPR category.

In TPR, dimensionality reduction methods decrease the dimension while ensuring that the topology of data is preserved. This category of methods is more powerful and more complex than DPR approaches. Two subcategories have been mentioned for TPR: **first**, data-independent approaches (which are based on predefined topology) such as *Self-Organizing Maps* [108], and **second**, data-dependent approaches (which topology is constructed based on data) such as LLE [159], and *Laplacian Eigenmaps* [17]. Primarily, data-dependent approaches outcome more robust results, but with the cost of higher complexity.

Several other NLDR approaches and variations have been proposed in the literature. Interested reader can refer to [118, 47, 112].

Appendix B

Binary Hashing

Here, one of the most well-known compact code approaches is discussed that can help speed up the process of nearest neighbour searching. A notable application of binary codes is in *binary hashing* [161] which has been the topic of significant research in the recent decade. The goal of binary hashing is to encode high-dimensional items, such as images, with *compact* binary strings subject to preserve a given notion of similarity. Such codes enable extremely fast *nearest neighbour search* as the distance between two codes (often the *Hamming distance*) can be computed quickly using bit-wise operations implemented at the hardware level.

As mentioned in Chapter 2, many binary hash algorithms have been proposed in the literature, such as iterative quantization [78], spectral hashing [192], graph hashing [128], supervised hashing [127], semi-supervised hashing [187], and minimal loss hashing [143].

The idea of hashing algorithms is to create a unique hash (signature or fingerprint) for each data point. If two data points are identical, hashing algorithms should return identical hash codes; if data points are even quite similar with minor differences, these algorithms will return completely different hash codes.

Despite that, in many applications, a concern is to find near duplicate or similar data points and not identical ones. In that case, hash functions that preserve similarity are needed.

Similarity-preserving hash functions need to be *correct* and *complete*. A hash function $h : X \mapsto Y$ is correct if:

$$\begin{aligned} &\forall x_1, x_2 \in X : \ if\ d_x(x_1, x_2) \leq \epsilon_x \\ &\exists \epsilon_y\ s.t\ d_y(h(x_1), h(x_2)) \leq \epsilon_y \end{aligned} \tag{B.1}$$

and is complete if:

$$\begin{aligned} &\forall x_1, x_2 \in X : d_y(h(x_1), h(x_2)) \leq \epsilon_y \\ &\exists \epsilon_x \ s.t \ \ if \ d_x(x_1, x_2) \leq \epsilon_x \end{aligned} \tag{B.2}$$

Another major concern may arise when one searches for similar items (of any kind). There may be far too many pairs of items to test for their degree of similarity, even if the similarity computing of any one pair can be done very easily. That concern motivates a technique called *Locality Sensitive Hashing (LSH)* for focusing the search on pairs that are most likely to be similar. Locality sensitive functions can apply to different spaces or different distance measures (such as Jaccard, Hamming, Euclidean distances). The SimHash LSH scheme has been selected in this research as LSH is one of the most well-known approaches for transferring data to Hamming space.

The general idea of SimHash LSH (SimHash [36] is used in this research) is that if there are similar data points, unique but similar hash codes are needed. Therefore, one can utilize those hash codes as a key for making hash tables repeatedly and randomly. These hash codes can be used for figuring out if data points are closely related without comparing them bit by bit. For the SimHash scheme [36], a collection of random vectors are given. The distance between any two vectors is the angular distance between them, $d(\mathbf{u}, \mathbf{v}) = \angle(\mathbf{u}, \mathbf{v})$. Consider the following hash functions used in SimHash LSH:

- **Projection**: compute the dot product of random vector $\mathbf{r}$ and $r \cdot u$
- **Rounding**: return $h_r(\mathbf{u}) = sign(\mathbf{u} \cdot \mathbf{v})$.

and the probability of different hash codes would be:

$$Pr[h(\mathbf{u}) \neq h(\mathbf{v})] = \angle(\mathbf{u}, \mathbf{v})/\pi \tag{B.3}$$

That is to say, to make hash codes, space is cut with random hyperplanes, and based on the position of each data point, bits are assigned to it. The computational complexity of SimHash for n points, d dimensions, k hyperplanes and r number of repetitions can be calculated as follows: dk is the cost of finding buckets, $n/2^k$ is the average number of points in each bucket, so the cost of comparison inside each bucket is $dn/2^k$. Thus, the total cost would be $rdk + rdn/2^k$. If k is in the order of $log(n)$, then SimHash is $O(log(n))$. Therefore, if the length of hash codes is fixed, the order of complexity will be in constant time. Arbitrary low false negative ($FN = (1-p^k)^r$) and false positive ($FP = 1-(1-q^k)^r$)

errors are achievable by manipulating the k and r variables, but it is sometimes quite expensive to do so.

More detailed discussion of approximate hashing by random projection is beyond the scope of this thesis and can be found in [36] and [12].

Appendix C

Evaluation Measures

Intra-cluster and inter-cluster similarity values can measure the quality of clustering algorithms. As these names indicate, intra-cluster similarity means how similar the items in clusters are, and the inter-cluster similarity is how similar the items between different clusters are. In addition to quality measures of clustering, there are less-intuitive evaluations for clustering, which are the resources –time and space (i.e. memory) – that a clustering approach needs to perform. The clustering time and space are not important issues in small datasets with current machines' power and memory, but they show their importance and severity when the size of a dataset grows.

One can categorize clustering evaluation measures into internal and external. As clustering is an exploratory data analysis task, there is no a priori knowledge about the parameters of cluster analysis. Therefore, the clustering objective functions try to optimize the internal clustering evaluation measures with no information about items except their feature values. In other words, the objective functions aim to minimize the intra-cluster similarity and maximize the inter-cluster similarity values just by using input features. This type of evaluation measure –which is based on feature values only– is called internal evaluation.

Internal Measures

Here, some data similarity measures are briefly described which are the foundations of *internal evaluation measures* and related to current research. These measures are either distance d or similarity Sim metrics between two variables $\mathbf{A}$ and $\mathbf{B}$. $\mathbf{A}$ and $\mathbf{B}$ can be either cluster representatives or objects(i.e. singleton clusters). Distance and similarity are related notions as the lower the distance, the higher the similarity and vice versa. As the current research is on attributes(features) in real and binary spaces, the focus will be on measures related to these data types. It is worth mentioning dissimilarity (distance) common properties in metric spaces:

$$\forall A, B : d(A,B) \geq 0 \text{ and } d(A,B) = 0 \Leftrightarrow A = B, \tag{C.1}$$

$$\forall A, B : d(A,B) = d(B,A), \tag{C.2}$$

$$\forall A, B, \text{ and } C : d(A,B) + d(B,C) \geq d(A,C) \tag{C.3}$$

In the given set of records $X = \{\mathbf{x}_i\}_{i=1}^{n}$, each item $\mathbf{x}_i$ is a vector so $\mathbf{x}_i = \{x_{i,k}\}_{k=1}^{dim}$. $\mathbf{x}_i$ and $\mathbf{x}_j$ can be either a cluster representative or a record. Those can be in the original space or dimensionality reduced space.

Real Space Measures: The original data in some parts of the current research is in real space. Therefore, It would be good to describe some real space measures here briefly.

Euclidean Distance: The Euclidean distance between a pair of items(i.e. ith and jth) is:

$$\forall \mathbf{x}_i, \mathbf{x}_j \in X : d_E(i,j) = (\sum_{k=1}^{dim} (x_{i,k} - x_{j,k})^2)^{1/2} \tag{C.4}$$

This measure can be tuned by using different weights on each feature which results *weighted Euclidean distance*:

$$\forall \mathbf{x}_i, \mathbf{x}_j \in X : d_{WE}(i,j) = (\sum_{k=1}^{dim} W_k (x_{i,k} - x_{j,k})^2)^{1/2} \tag{C.5}$$

The general form of Euclidean distance is *Minkowski distance*

$$\forall \mathbf{x}_i, \mathbf{x}_j \in X \text{ and } m \geq 1 : d_M(i,j) = (\sum_{k=1}^{dim} (x_{i,k} - x_{j,k})^m)^{1/m} \tag{C.6}$$

Euclidean distance (which is also called L_2 norm), *Manhattan* or *city block distance*(which is also called L_1 norm), and *Chebyshev* or *supremum distance*(i.e. L_∞ norm) can be derived from Minkowski distance. It is worth mentioning that all mentioned distances are metric. Here, Minkowski family of distance function is mentioned in Table C.1. For a more comprehensive study one can go to reference [35].

Measure	**Definition**
Manhattan or City Block	$d_{CB}(i,j) = \sum_{k=1}^{dim} \lvert x_{i,k} - x_{j,k}) \rvert$
Euclidean	$d_E(i,j) = (\sum_{k=1}^{dim}(x_{i,k} - x_{j,k})^2)^{1/2}$
weighted Euclidean distance	$d_{WE}(i,j) = (\sum_{k=1}^{dim} W_k(x_{i,k} - x_{j,k})^2)^{1/2}$
Minkowski	$d_M(i,j) = (\sum_{k=1}^{dim}(x_{i,k} - x_{j,k})^m)^{1/m}$
Chebyshev	$d_C(i,j) = (\sum_{k=1}^{dim}(x_{i,k} - x_{j,k})^\infty)^{1/\infty}$

Table C.1: Minkowski Family of Dissimilarity Measures in Real Space

Binary Space Measures: The target space in current research is a short code binary space. Here, the binary space measures are briefly discussed which can also be categorized as internal measures of clustering.

In the given set of binary records $\mathcal{B} = \{\mathbf{b}_i\}_{i=1}^n$, each item $\mathbf{b}_i$ is a vector so $\mathbf{b}_i = \{b_{i,k}\}_{k=1}^{dim}$ and $b_{i,k} \in \{0,1\}$. $\mathbf{b}_i$ and $\mathbf{b}_j$ can be either a cluster representative or a record. Those can be in the original space- if the original space is binary space- or in dimensionality reduced space.

$$\forall \mathbf{b}_i, \mathbf{b}_j \in \mathcal{B} : |\mathbf{b}_i \cap \mathbf{b}_j|_{k=1}^{dim} = \sum_{k=1}^{dim}(b_{i,k} = b_{j,k} = 1) = \sum_{k=1}^{dim}(b_{i,k} \wedge b_{j,k}) = \mathbf{b}_i \cdot \mathbf{b}_j \tag{C.7}$$

$$\forall \mathbf{b}_i, \mathbf{b}_j \in \mathcal{B} : | \sim \mathbf{b}_i \cap \sim \mathbf{b}_j|_{k=1}^{dim} = \sum_{k=1}^{dim}(b_{i,k} = b_{j,k} = 0) = \sum_{k=1}^{dim}(\sim b_{i,k} \wedge \sim b_{j,k}) = \sim \mathbf{b}_i \cdot \sim \mathbf{b}_j \tag{C.8}$$

$$\forall \mathbf{b}_i, \mathbf{b}_j \in \mathcal{B} : | \sim \mathbf{b}_i \cap \mathbf{b}_j|_{k=1}^{dim} = \sum_{k=1}^{dim}(b_{i,k} = 0, b_{j,k} = 1) = \sum_{k=1}^{dim}(\sim b_{i,k} \wedge b_{j,k}) = \sim \mathbf{b}_i \cdot \mathbf{b}_j \tag{C.9}$$

$$\forall \mathbf{b}_i, \mathbf{b}_j \in \mathcal{B} : |\mathbf{b}_i \cap \sim \mathbf{b}_j|_{k=1}^{dim} = \sum_{k=1}^{dim}(b_{i,k} = 1, b_{j,k} = 0) = \sum_{k=1}^{dim}(b_{i,k} \wedge \sim b_{j,k}) = \mathbf{b}_i \cdot \sim \mathbf{b}_j \tag{C.10}$$

All binary measures can be derived from the combination of these four quantities. In order to make the notation simple, TT, FF, FT, and TF are used equivalent to previous equations respectively.

Hamming Distance: The target space in which the proposed clustering algorithm is implemented is a compressed binary space. Therefore, Hamming distance is utilized as an internal criterion for cluster similarity measure. This measure can be defined by the following equation:

$$\forall \mathbf{b}_i, \mathbf{b}_j \in X : d_{Ham}(i, j) = TF + FT \tag{C.11}$$

Which means the Hamming distance is the one population count on $\mathbf{b}_i$ XOR $\mathbf{b}_j$. There are several other binary distance measures which can be used as an internal criterion of clustering evaluation which will be mentioned by C.3 and C.2. For a more comprehensive study on binary space similarity and dissimilarity measures one can go to reference "A Survey of Binary Similarity and Distance Measures [42]".

Hamming distance is one of the most straightforward and widely used distance measures in binary space. For a finite length of the binary code, Hamming distance can be calculated simply by a look-up table which can decrease the computational complexity of distance finding to constant time. There are some other distance measures with the same simplicity, but they can be derived from each other as they belong to the same family of distance functions. Therefore, in the current research, Hamming distance is utilized for intra-cluster and inter-cluster similarity measure in binary space.

External Measures

As mentioned, cluster evaluation includes two variations: internal and external measures. Internal measures give negative points to inter-cluster similarities and positive points to intra-cluster similarities. One can find examples in which high scores of internal measures for cluster evaluation do not necessarily mean good clustering performance. On the other hand, *external clustering measures* analyze the closeness of clustering to some references (standard). External clustering measures come into account when clustering is done, and an external audit is needed to verify the performance of the clustering approach.

An example, *Purity*, is one of the simplest external evaluation methods. Each cluster is assigned to the largest amount of class labels in that cluster. Based on the definition 5, having a set of clusters, C, including i clusters, and a set of classes, G, including j classes,

Measure	**Definition**
Jaccard	$\frac{TT}{TT+FT+TF}$
Simple matching coefficient of Sokal & Michener	$\frac{TT+FF}{TT+FT+TF+FF}$
Sokal & Sneath	$\frac{TT}{TT+2(FT+TF)}$
Rogers & Tanimoto	$\frac{TT+FF}{TT+2(FT+TF)+FF}$
Dice or Sorensen	$\frac{2TT}{2TT+FT+TF}$
Hamann	$\frac{TT-(FT+TF)+FF}{TT+FT+TF+FF}$
Ochiai	$\frac{TT}{\sqrt{(TT+FT)(TT+TF)}}$
Sokal & Sneath	$\frac{TTFF}{\sqrt{(TT+FT)(TT+TF)(FF+FT)(FF+TF)}}$
Phi of Pearson	$\frac{TTFF-FTTF}{\sqrt{(TT+FT)(TT+TF)(FF+FT)(FF+TF)}}$
Gower & Legendre	$\frac{TT}{TT+FT+TF+FF}$
Cosine	$\frac{TT}{\sqrt{(TT+FT)(TT+TF)}^2}$

Table C.2: Some Similarity Measures For Binary Data

Measure	**Definition**
Hamming	$TF + FT$
Euclidean	$\sqrt{TT + FT + TF}$
Squared Euclidean	$\sqrt{TT + FT + TF}^2$
Manhattan or City Block	$FT + TF$
Mean-Manhattan	$\frac{TF+FT}{TT+FT+TF+FF}$
Minkowski	$(TF + FT)^{\frac{1}{m}}$
Lance & Williams	$\frac{TF+FT}{2TT+FT+TF}$

Table C.3: Some Dissimilarity Measures For Binary Data

the purity is calculated by

$$Purity(C, G) = \frac{1}{N} \sum_i max_j |G_i \cap C_j| \tag{C.12}$$

Purity lies between [0,1] when 0 shows no agreements between gold standard clustering results and clustering results and 1 shows complete agreement between the results and gold standard results.

Researchers tend to develop an intuition about theoretical judgments for external clustering evaluation, but the size of datasets, the variation of data types, and consequent diversified clustering algorithms have forced researchers to introduce various evaluation measures. Therefore, several schemes for clustering evaluation in different fields of data mining, statistics, and information retrieval have been proposed. Basically, these several measures can be classified into three main groups: *pair-counting*, *set-matching*, and *information and entropy-based* . In the current research, the pair-counting approaches are utilized. Therefore, in next sections definitions of the external clustering evaluation measures and some of their specifications (specifically for pair-counting approaches) are provided that help through the selection process. As the detailed discussion about other groups of measures is beyond the scope of this research, they are briefly mentioned them for reference.

Definition 5. *Given a set of records* $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^n$*, a set of clustering results* $\mathcal{G} = \{\boldsymbol{g}_j\}_{j=1}^k$

where each $\boldsymbol{g}_j$ is a set of records with gold standard label j, a set of clustering results $\mathcal{C} = \{\boldsymbol{c}_l\}_{l=1}^{m}$ where each $\boldsymbol{c}_l$ is a set of records with the output of clustering algorithm result labels l that needs to be evaluated, k and l are the number of clusters from gold standard labeling and from the clustering algorithm, respectively.

$$\sum_{j=1}^{k} \mathbf{g}_j = \sum_{l=1}^{m} \mathbf{c}_l = n \tag{C.13}$$

Definition 6. *A contingency table for previous definition (5) is a table with matrix $N_{k\times m} = [n_{pq}]$ format, where its margins are $\mathcal{G}$ and $\mathcal{C}$ and $n_{pq} = |\boldsymbol{g}_p \cap \boldsymbol{c}_q|$ and $p \in \{1, ..., k\}$ and $q \in \{1, ..., m\}$*

Therefore, the contingency table will have the following conditions:

$$\forall p \in \{1, ..., k\} \text{ and } q \in \{1, ..., m\} : n_{pq} \geq 0 \tag{C.14}$$

$$\forall q \in \{1, ..., m\} : \sum_{p=1}^{k} n_{pq} = |\mathbf{g}_q| = b_q \tag{C.15}$$

$$\forall p \in \{1, ..., k\} : \sum_{q=1}^{m} n_{pq} = |\mathbf{c}_p| = a_p \tag{C.16}$$

$$\begin{array}{c|cccc|c} C\diagdown^{G} & g_1 & g_2 & \dots & g_k & \text{Sums} \\ \hline c_1 & n_{11} & n_{12} & \dots & n_{1k} & a_1 \\ c_2 & n_{21} & n_{22} & \dots & n_{2k} & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_m & n_{m1} & n_{m2} & \dots & n_{mk} & a_m \\ \hline \text{Sums} & b_1 & b_2 & \dots & b_k & \end{array} \tag{C.17}$$

In order to simplify describing some of evaluation measures, another contingency table can be defined:

Definition 7. *For the given set of records $\mathcal{X}$, the set of gold standard labels $\mathcal{G}$, and the set of clustering results $\mathcal{C}$;* ***true positive*** *is the number of records that are in the same sets in $\mathcal{G}$ and in the same sets in $\mathcal{C}$;* ***true negative*** *is the number of records that are in the different sets in $\mathcal{G}$ and the different sets in $\mathcal{C}$;* ***false positive*** *is the number of records that are in the different sets in $\mathcal{G}$ but in the same sets in $\mathcal{C}$;* ***false negative*** *is the number of records that are in the same sets in $\mathcal{G}$ but in the different sets in $\mathcal{C}$:*

		C Pair of Records in The Same Cluster	Pair of Records in Different Clusters
G	Pair of Records in The Same Cluster	TP	FN
	Pair of Records in Different Clusters	FP	TN

Table C.4: 2×2 Contingency Table

$$TP = |\{(x_i, x_j)|x_i, x_j \in g_p \wedge x_i, x_j \in c_q\}| \tag{C.18}$$

$$\begin{aligned} TN = |\{(x_i, x_j)|x_i \in g_l, x_j \in g_p \wedge x_i \in c_t, x_j \in c_q, l \neq p \wedge t \neq q\}| = \\ |\{(x_i, x_j)|\nexists p, q : x_i, x_j \in g_p \wedge x_i, x_j \in c_q\}| \end{aligned} \tag{C.19}$$

$$\begin{aligned} FP = |\{(x_i, x_j)|x_i \in g_l, x_j \in g_p \wedge x_i, x_j \in c_q, l \neq p\}| = \\ |\{(x_i, x_j)|\nexists p : x_i, x_j \in g_p \wedge x_i, x_j \in c_q\}| \end{aligned} \tag{C.20}$$

$$\begin{aligned} FN = |\{(x_i, x_j)|x_i, x_j \in g_p \wedge x_i \in c_t, x_j \in c_q, t \neq q\}| = \\ |\{(x_i, x_j)|\nexists q : x_i, x_j \in g_p \wedge x_i, x_j \in c_q\}| \end{aligned} \tag{C.21}$$

This definition leads to the smaller (i.e 2×2) contingency table (Table C.4).

In order to find values of 2×2 contingency table, one can use the $k \times m$ contingency table, which is defined in definition 6:

$$TP = \sum_{p=1}^{k} \sum_{q=1}^{m} \binom{d_{pq}}{2} \tag{C.22}$$

$$FN = \sum_{p=1}^{k} \binom{d_{p.}}{2} - TP \tag{C.23}$$

$$FP = \sum_{q=1}^{m} \binom{d_{.q}}{2} - TP \tag{C.24}$$

$$TN = \binom{n}{2} - TP - FN - FP \tag{C.25}$$

The literature uses the values from the 2×2 contingency table in several pair-counting clustering evaluations, but here only certain famous ones are mentioned:

$$Precision = \frac{TP}{TP + FP} \quad , Recall = \frac{TP}{TP + FN} \tag{C.26}$$

$$F - Measure = \frac{2TP}{2TP + FP + FN} \tag{C.27}$$

$$RandIndex = \frac{TP + TN}{TP + FP + FN + TN} \tag{C.28}$$

$$Jaccard = \frac{TP}{TP + FP + FN} \tag{C.29}$$

Rand Index (RI) is one of the most-popular pair-counting measures for counting pairs of items on which two clustering results (i.e., gold standard clustering, and the clustering algorithm that needs to be evaluated) agree or disagree. This well-known index lies in the nominal range of [0,1] when 0 shows no agreements between data labels (i.e. gold standard clustering results) and clustering results, and 1 shows complete agreement between results and class labels. However, one can notice that this index often lies between a smaller range of [0.5,1]. Therefore, another form of RI has been proposed and used, which is the *Adjusted Rand Index (ARI)* [91].:

$$\overbrace{ARI}^{\text{Adjusted Index}} = \frac{\overbrace{\sum_{ij}\binom{n_{ij}}{2}}^{\text{Index}} - \overbrace{[\sum_{i}\binom{a_i}{2}\sum_{j}\binom{b_j}{2}]/\binom{n}{2}}^{\text{Expected Index}}}{\underbrace{\frac{1}{2}[\sum_{i}\binom{a_i}{2} + \sum_{j}\binom{b_j}{2}]}_{\text{Max Index}} - \underbrace{[\sum_{i}\binom{a_i}{2}\sum_{j}\binom{b_j}{2}]/\binom{n}{2}}_{\text{Expected Index}}} \tag{C.30}$$

n_{ij}, a_i, and b_j are values from a contingency table (Equation C.17).

Apart from these measures, there are many less-popular measures in the pair-counting measure class. For a comprehensive list of pair-counting measures, one can use a study by Albatineh et al. [8], which narrows down 28 measures to 22 different ones, or another study by Warrens [191]. These lists are large enough to make measure selection a confusing task

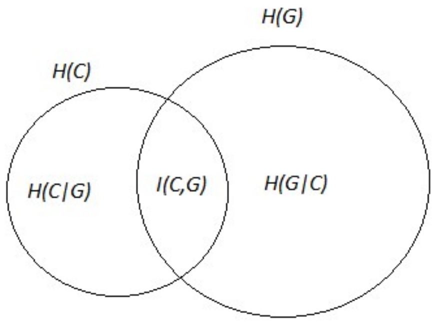


Figure C.1: Information Theoretic Quantities

of clustering algorithm evaluation. Regardless of this considerable number of measures, RI and ARI are the most well-known and used ones. Therefore, in this research RI and ARI are used as one part of the clustering evaluation. In the next paragraphs, a very brief description of two other families of external measures is given.

Set-matching-based measures are another class of the external measures used for cluster validity. These measures are based on finding matches between clusters, in different clusterings (which in the current definition are G and C). There are two drawbacks to this class of measures, both of which originate from the "problem of matching": (i) different numbers of clusters for G and C, and (ii) ignoring what happens to the unmatched part of each cluster. One can find a more thorough discussion on this class of indices (i.e. measures) in [87]. Purity, an example of the set-matching class of measures, has been introduced in preceding paragraphs. Purity is also utilized as a validation measure in the current research.

The other class of measures is comprised of *information theoretic-based measures* which is based on the fundamentals of information theory, in which clustering is reviewed as random variables with a joint distribution and the mutual information between two clusterings must be judged. The idea is to know how much information is in each of the clusterings and how much information one clustering has about the other. Figure C.1 shows information theoretic quantities. $H(C)$ represents Entropy related to clustering C (which is always a non-negative number) is the measure of the amount of randomness of the variable associated with C. The mutual information $I(C, G)$ is how much information C (i.e. a proposed clustering) has about G (i.e. gold standard clustering). $H(C|G)$ and $H(G|C)$ are conditional entropy. Several measures has been proposed in this class, such as *conditional entropy*, *joint entropy*, *mutual information*, various *normalized mutual information*, *varia-*

tion of information, and etc. For more details about this class of measures, the reader is invited to consult [134].

No one can claim that there is a clustering algorithm which is the best clustering method out there and fits every clustering problem. Likewise, no one can claim that there is an external clustering validity measure which is the best measure for clustering comparison and is optimal for every problem. Yet, one can find some properties which help external measure selection for clustering validity which are *Metric Properties*, *Normalization* (Range), *Sample Size Dependency*, and *coarsening and Refining clusters Dependency.* More discussion about these properties is outside the scope of this research which can be found in clustering literature.

As mentioned, in this research, RI, ARI, and purity are used which are the most well-known and widely used measures to validate clustering algorithms.

Summary

In this chapter, some external and internal validity (evaluation) measures are introduced that can be summarized as follows:

i) External validity indices are the measures of the agreement between two partitions, one of which is usually a known/golden partition, e.g. true class labels, and another is from the clustering procedure. *Purity*, *Rand index*, *adjusted Rand index*, *Jaccard index*, and *Fowlkes-Mallows (FM) index* are some examples of this category.

ii) Internal validity indices evaluate clustering results by using only features and information inherent in a dataset. They are usually used in the case that true solutions are unknown. *Silhouette index*, *Davies-Bouldin*, *Calinski-Harabasz*, *Dunn index*, *R-squared index* are some examples of internal validity measures.