B.TECH. PROJECT REPORT

On

Approximate k-NN Graph in Chameleon Clustering

By

M. Sai Akshay Reddy : 200001049

&

Rishi Parsai : 200001068

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

November, 2023

Approximate k-NN Graph in Chameleon Clustering

A PROJECT REPORT

Submitted in partial fulfillment of the
requirements for the award of the degrees

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

M. Sai Akshay Reddy : 200001049

&

Rishi Parsai : 200001068

Discipline of Computer Science and Engineering

Indian Institute of Technology Indore

Guided by:

Prof. Kapil Ahuja

Professor

Computer Science and Engineering

IIT Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE

November, 2023

CANDIDATES' DECLARATION

We hereby declare that the project entitled **"Approximate k-NN Graph in Chameleon Clustering"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Computer Science and Engineering' completed under the supervision of **Dr. Kapil Ahuja, Professor, Computer Science and Engineering**, IIT Indore is an authentic work.

Further, we declare that we have not submitted this work for the award of any other degree elsewhere.

AKSHAY REDDY
20/11/2023

M. Sai Akshay Reddy **Rishi Parsai**

CERTIFICATE by BTP Guide

It is certified that the above statement made by the student is correct to the best of my knowledge.

20/11/2023

Dr. Kapil Ahuja
Professor
Discipline of Computer Science and Engineering
IIT Indore

PREFACE

This report on "Approximate k-NN Graph in Chameleon Clustering" is prepared under the guidance of Dr. Kapil Ahuja, Professor, Computer Science and Engineering, IIT Indore.

In this report, we employ a range of approximate nearest neighbor algorithms to construct the K-nearest neighbor graph within the context of the Chameleon Clustering Algorithm. Our goal is to realize substantial performance enhancements compared to the conventional Chameleon Clustering Algorithm. To validate our approach, we have conducted experiments using the 32 benchmark datasets as originally provided by the algorithm's authors. A comprehensive exposition of these approximate nearest neighbor algorithms is presented in this report..

We attempted to provide an explanation for the experiments we conducted using a variety of feature extraction (and reduction) methods. Subsequently, we discussed how we experimented with various combinations of these methods and compared the results.

ACKNOWLEDGEMENTS

We want to thank our B.Tech Project supervisor **Dr. Kapil Ahuja** for his guidance and constant support in structuring the project and providing valuable feedback throughout the course of this project. His overseeing the project meant there was a lot that we learnt while working on it. We thank him for his time and efforts.

We are grateful to **Mr. Priyanshu Singh**, without whom this project would have been impossible. He provided valuable guidance to handle the delicacies involved in the project and also taught us how to write a scientific paper.

We are grateful to the Institute for the opportunity to be exposed to systemic research, especially Prof. Kapil Ahuja's Lab, for providing the necessary hardware utilities to complete the project.

Lastly, we offer our sincere thanks to everyone who helped us complete this project, whose name we might have forgotten to mention.

M. Sai Akshay Reddy & Rishi Parsai
B.Tech. 4^{th} **Year**
Discipline of Computer Science and Engineering
IIT Indore

ABSTRACT

Conventional clustering algorithms often fall short in delivering human-like results when dealing with data exhibiting varying densities, intricate distributions, or the presence of noise. In response to this, we introduce an enhanced graph-based clustering method named Chameleon 2, which effectively addresses several limitations observed in current clustering techniques. Our enhancements include modifications to the internal cluster quality assessment and the incorporation of an additional step to fortify algorithm resilience. Our findings demonstrate a noteworthy enhancement in clustering quality, as assessed through Normalized Mutual Information, across 32 synthetic datasets widely employed in clustering research. Furthermore, this substantial improvement is consistently validated when applied to real-world datasets.

Approximate Nearest Neighbor Search (ANNS) stands as a pivotal and indispensable operation in a multitude of domains, including databases, machine learning, multimedia, and computer vision. While numerous algorithms are consistently introduced in these domains every year, a comprehensive assessment and scrutiny of their performance has been notably lacking in the literature.

We employ different Approximate Nearest Neighbor Algorithms like FLANN, HNSW in the Chameleon 2 Clustering Algorithm and validate them over the 32 benchmark datasets. The average Normalised Mutual Info Score over these 32 datasets for FLANN is 0.905 and for HNSW is 0.929 .

Contents

List of Tables

List of Figures

List of Abbreviations

Ch	**Ch**ameleon
Ch2	**Ch**ameleon **2**
KNN	**K** **N**earest **N**eighbors
ANN	**A**pproximate **N**earest **N**eighbors
FLANN	**F**ast **L**ibrary **A**pproximate **N**earest **N**eighbors
HNSW	**H**ierarchical **N**avigable **S**mall **World**
ML	**M**achine **L**earning

Chapter 1

Introduction

1.1 Background

The classification or grouping of objects is a necessity across various domains, including engineering, scientific disciplines, marketing, technology, humanities, medical sciences, and everyday activities.

For example, in the realm of marketing, the practice of categorizing customers based on their shopping behavior has become an invaluable strategy. By classifying customers with similar preferences and habits, businesses can craft more personalized and targeted marketing campaigns. For instance, customers who consistently purchase organic and health-focused products may be labeled as "wellness-conscious," while those who frequently buy the latest gadgets and electronics might fall into the "tech-savvy" category. This categorization enables businesses to create tailored promotional offers, product recommendations, and advertising messages that resonate with each specific group. By doing so, companies can significantly improve their customer engagement, conversion rates, and overall sales performance. However, it is essential for marketers to accurately identify and understand these customer preferences and behaviors. Misclassification can lead to ineffective marketing strategies and could even alienate potential customers. Therefore, ensuring the accurate labeling of customers is a fundamental aspect of successfully leveraging the power of customer categorization in marketing.

Clustering involves a process of exploration, where data points or objects are organized into clusters in a way that maximizes similarity within each cluster while minimizing similarity between different clusters. The main objective of clustering is to uncover underlying patterns, structures, or inherent groupings within a dataset without any prior information about these groupings.

1.2 Previous Work

Clustering Algorithms like DBSCAN [1], K-means [2], ROCK [3], CURE [4] are primarily designed to identify clusters that conform to fixed, predetermined models. For example, DBSCAN presupposes all points inside authentic clusters show density connectivity, while points from distinct clusters lack this connection. K-means algorithm assumes that clusters are of globular shape.

Algorithms like CURE, ROCK depend on a fixed model to determine which clusters to combine during the clustering procedure. CURE decides if two clusters are alike by checking how similar their closest representative points are, without considering how close the points inside each cluster are, whereasROCK figures out if two clusters are similar by comparing how they are connected overall, using a predefined model for connectivity.

These methods might face problems when the chosen settings in the fixed model don't match well with the dataset when grouping or if the model is not good enough to understand the real features of the clusters. Furthermore, many of these methods face challenges when used on data that includes clusters with different shapes, densities, and sizes.

So to overcome these problems, we are going to use Chameleon Clustering Algorithm and Chameleon 2 Clustering Algorithm which compares two clusters by using a dynamic model.

1.3 Objectives

Numerous clustering algorithms have been introduced over time. Some of these algorithms are grounded in density-based principles [5],[4]. Others leverage graph-based representations [6] while certain approaches involve fitting models to data distributions. [7], [8].

In the chameleon clustering algorithm, creating the KNN graph with the brute force method is slow and the most costly part. To solve this issue, we use approximate nearest

neighbor algorithms to build the K nearest neighbor graph, trading a bit of accuracy for efficiency, especially beneficial for large datasets.

The rest of the report is structured in the following manner - Chapter 2 briefly describes about the Chameleon and Chameleon 2 Clustering Algorithms and the drawbacks of them. Chapter 3 presents the detailed explanation of approximate nearest neighbor alogrithms we've used in our project namely FLANN and HNSW. Chapter 4 gives the experimental results. In Chapter 5 we present the conclusions and the scope for the future work.

Chapter 2

Literature Review

In this chapter, we're going to talk about some of the research we've looked at to help us decide how to go about our project and put our ideas into action.

As we dug into various research papers, we discovered that the Chameleon and Chameleon 2 clustering algorithms are very important in the area of machine learning and computer vision. These algorithms are like special tools that help computers understand and group data, making them really important in these areas. We'll explore how these algorithms work and why they matter in the upcoming sections, and we'll also discuss how they relate to our own project goals.

2.1 Chameleon Algorithm

In this part, we'll delve into the CHAMELEON Algorithm, a new way of clustering that aims to fix the issues found in existing clustering methods.

CHAMELEON works on a scattered graph, where nodes represent data items, and the weights of the edges indicate the similarity between these data items. CHAMELEON identifies clusters within the dataset through a two-step process. In the first step, it uses a method to divide the data items into several smaller groups. In the next step, it employs a step-by-step approach to combine these smaller groups together and identify the real clusters.

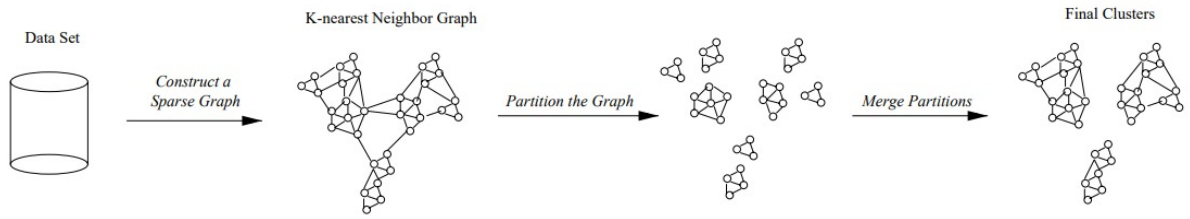

FIGURE 2.1: Chameleon Algorithm Steps[9]

The crucial feature of CHAMELEON algorithm is its capability to find the most similar pairs of clusters by taking into account both their inter-connectivity and closeness. In doing so, it effectively addresses the limitations which arise from the utilization of only one of these factors. Moreover, CHAMELEON presents a new method to represent how much clusters are connected and close to each other, considering the specific features of the clusters. This implies that CHAMELEON doesn't depend on a fixed model given by the user; rather, it can adjust automatically to the unique characteristics of the merging clusters.

- The initial step includes constructing a graph using the idea of k nearest neighbors. Every node is connected to its k closest neighbors through an edge, and the weight of this edge is the inverse of the distance between them.

- In the second stage, we divide the graph that was created earlier. The goal of this step is to create partitions that are approximately the same size while reducing the number of edges that need to be cut. Consequently, numerous small clusters emerge, with each having a handful of nodes that maintain strong connections within their specific partitions. For the partitioning process; [10], [11] use their custom algorithm, **hMETIS**. This algorithm operates across various levels of the graph, starting with a coarsened version.

- The last and most crucial stage is to combine the divided clusters using Chameleon's dynamic modelling framework. To do this, start by combining the clusters that are the most similar to each other. What distinguishes Chameleon is the earlier partitioning phase, which enables us to merge clusters comprising numerous objects. Consequently, the number of merge operations required is substantially reduced compared to traditional hierarchical clustering methods. Chameleon employs the average-link method to assess the similarity between clusters. Furthermore, in addition to calculating the distances between clusters, it also takes into account the distances within each cluster during the merging process.

2.1.1 Constructing K-Nearest Neighbor Graph

Every vertex in the KNN graph represents a data item, and if two vertices represent data items that are among the k most similar to each another, then an edge exists between them.

Using a k-nearest neighbour graph to describe data has various benefits such as,

- To begin with, nodes that are distant from each other are not connected.
- The graph records the density of different areas by assigning weights to the edges. In this context, edges connecting data points in dense regions of the graph tend to have higher weights, signifying stronger similarities, while edges in sparse regions tend to have lower weights. As a result, when the graph is divided into two parts using a min-cut approach, it effectively delineates the boundary between the sparse region and the rest of the graph.

Finally, the graph offers a computational benefit in various graph-based algorithms compared to a complete graph. This advantage is particularly evident in algorithms that involve tasks such as graph partitioning and refining partitions.

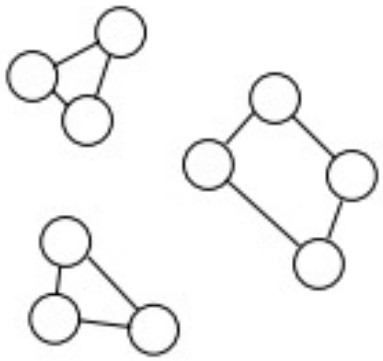

FIGURE 2.2: 2-NN graph [9]

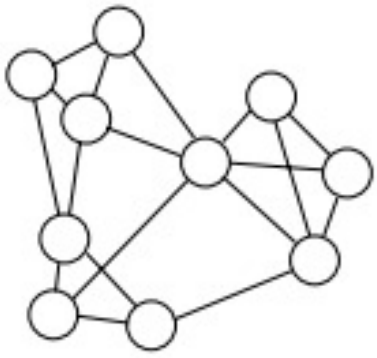

FIGURE 2.3: 3-NN graph [9]

2.1.2 Partitioning the K-Nearest Neighbor Graph

Using a graph partitioning algorithm, CHAMELEON splits the KNN graph into multiple partitions in order to identify sub-clusters. The goal is to minimize the edgecut, which is the total weight of edges connecting different partitions. Since each edge in the KNN graph represents the similarity between nodes, a partitioning strategy that minimizes the edge-cut helps decrease the connections among data points in the resulting partitions.

In particular, CHAMELEON leverages the partitioning algorithm within the hMETIS library. [12]. Previous studies [13], [10], [14] have demonstrated that hMETIS is capable of rapidly generating high-quality partitionings for diverse unstructured graphs.

In CHAMELEON, we mainly use hMETIS to split a cluster into two sub-clusters while minimizing the connections between them. Each of these sub-clusters must also include at least 25% of the nodes from the original cluster. The CHAMELEON algorithm establishes the first group of smaller clusters through the following process:

- It starts off with every point grouped into a single cluster. The largest sub-cluster within the existing sub-clusters is then iteratively determined, and hMETIS is utilised to carry out a bisecting operation.
- This process comes to an end when the largezt sub-cluster has fewer vertices than a predetermined threshold, designated as MINSIZE. Essentially, MINSIZE ought to be set to a value that is less than the size of the majority of clusters that the

dataset is expected to contain. Simultaneously, MINSIZE must be large enough to ensure that most sub-clusters encompass an adequate number of nodes.

Typically, for the datasets we encountered, a successful approach was to set MINSIZE at approximately 1-5% of the size of the dataset.

2.1.3 Merging using dynamic modeling

Once the partitioing phase is done, CHAMELEON then switches to an agglomerative hierarchical clustering approach which involves merging the small sub-clusters produced in the partitioning phase. The main objective is to identify the sub-clusters that exhibit the highest degree of similarity.

CHAMELEON uses a dynamic modeling framework to identify the most similar pair of clusters, taking into account both Relative Inter-Connectivity and Relative Closeness.

Relative Inter-connectivity : The ratio of internal interconnectivity of C_i and C_j to absolute interconnectivity between two clusters is known as relative interconnectivity.

Absolute inter-connectivity is described as the sum of edge weights connecting clusters C_i and C_j. This is denoted as E $EC(C_i, C_j)$. On the other hand, the internal interconnectivity of a cluster C_i is effectively represented by the size of its min-cut bisector, denoted as $EC(C_i)$. Therefore the relative inter-connectivity $RI(C_i, C_j)$ between C_i and C_j is equal to

$$RI(C_i, C_j) = \frac{|EC(C_i, C_j)|}{\frac{|EC(C_i)| + |EC(C_j)|}{2}}$$

Relative Closeness : The ratio of internal closeness of C_i and C_j to absolute closeness between two clusters is known as relative closeness.

The average weight of the edges joining the vertices in C_i to the vertices in C_j determines the Absolute Closeness. On the other hand, By calculating the average weight of the edges inside the cluster, internal closeness is evaluated. Therefore the relative closeness $RC(C_i, C_j)$ between C_i and C_j is equal to

$$RC(C_i, C_j) = \frac{\overline{S_{EC}}(C_i, C_j)}{\frac{|C_i|}{|C_i| + |C_j|}\overline{S_{EC}}(C_i) + \frac{C_j}{|C_i| + |C_j|}\overline{S_{EC}}(C_j)}$$

where the average weights of the edges that belong in the min-cut bisector of clusters C_i and C_j are denoted by $\overline{S_{EC}}(C_i)$ and $\overline{S_{EC}}(C_j)$, respectively, and the average weight of the edges connecting vertices in C_i to vertices in C_j is denoted by $\overline{S_{EC}}(C_i, C_j)$.

Several approaches exist to create an agglomerative hierarchical clustering algorithm that considers both of these metrics. CHAMELEON has incorporated two distinct schemes to address this.

- The initial scheme involves merging only those pairs of clusters where both the relative inter-connectivity and relative closeness surpass certain user-defined thresholds. These are denoted by T_{RI} and T_{RC}. In this method, CHAMELEON examines each cluster C_i and verifies whether any of its neighboring clusters C_j meet the following two criteria:

$$RC(C_i, C_j) \geq T_{RC} \, , \, RI(C_i, C_j) \geq T_{RI}$$

- If multiple neighboring clusters fulfill the mentioned conditions, CHAMELEON chooses to merge C_i with the cluster C_j that has the highest absolute inter-connectivity between them. After each cluster gets a chance to merge with one of its neighboring clusters, the chosen combinations are carried out, and the whole process repeats. The parameters T_{RI} and T_{RC} control the characteristics of the clusters we want to achieve. Based on the selected values for T_{RI} and T_{RC}, CHAMELEON's merging process may encounter a situation where it cannot continue because none of the neighboring clusters meet the specified conditions. Now either we can end the process and present the current clustering as the result or we can try to combine more pairs of clusters by gradually adjusting the two parameters, potentially at varying rates.

- In CHAMELEON's second method, a function combines the relative interconnectivity and relative closeness. The algorithm then chooses the pair of clusters that maximizes this function so that they can be merged. As our goal is to combine pairs that have both a high degree of inter-connectivity and a high degree of closeness, a natural way to define such a function is to maximize $RI(C_i, C_j) * RC(C_i, C_j)$. However, there are instances where we may prefer clusters that prioritize one of

these measures over the other. Hence, CHAMELEON Selects the combination of clusters that maximises the criteria,

$$RI(C_i, C_j)*RC(C_i, C_j)^{\alpha}$$

where α is a user-defined parameter. If α is greater than 1, CHAMELEON places greater emphasis on the relative closeness, while if α is less than 1, it prioritizes the relative inter-connectivity.

2.2 Drawbacks of Chameleon

Although Chameleon is regarded as one of the best clustering algorithms [15], our tests point out certain issues with the clustering that is produced. The primary limitation of the algorithm is its incapacity to manage clusters that encompass single nodes.

A common problem resulting from hMETIS partitioning is shown in Figure 2.4, where numerous disconnected clusters form in low-density areas. The final outcome includes quite a few clusters consisting of only one data point, and it is more appropriate to merge them with neighboring clusters rather than treating them as separate entities. This issue is not limited to partitioning; it also suggests that the similarity metric is not able to adequately represent the concept of a point's neighbourhood.

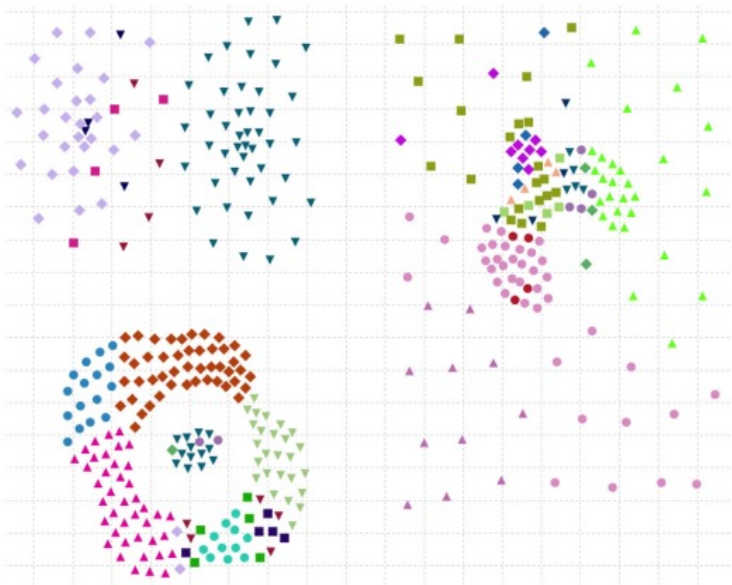

FIGURE 2.4: Quality of Ch1 Clustering is poor and it fails to identify singleton clusters [16]

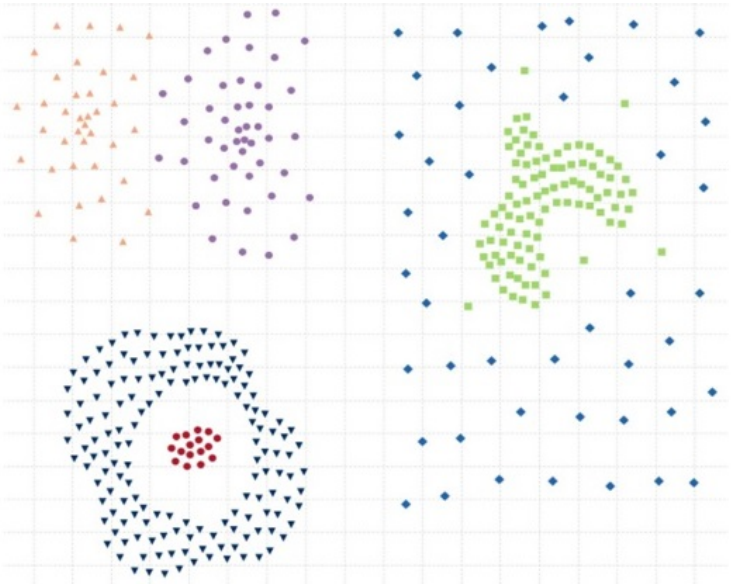

FIGURE 2.5: Standard Ch2 with recursive bisection partitioning [16]

2.3 Introduction to Chameleon 2 (Ch2) Clustering Algorithm

Adjusting the Chameleon algorithm can be tricky, as shown by various experiments. In addition to tweaking α, we also need to fine-tune several parameters of hMETIS to suit the specific dataset. As a result, we present an improved version of the Chameleon algorithm, called Chameleon 2 (Ch2). The specific modifications are detailed below.

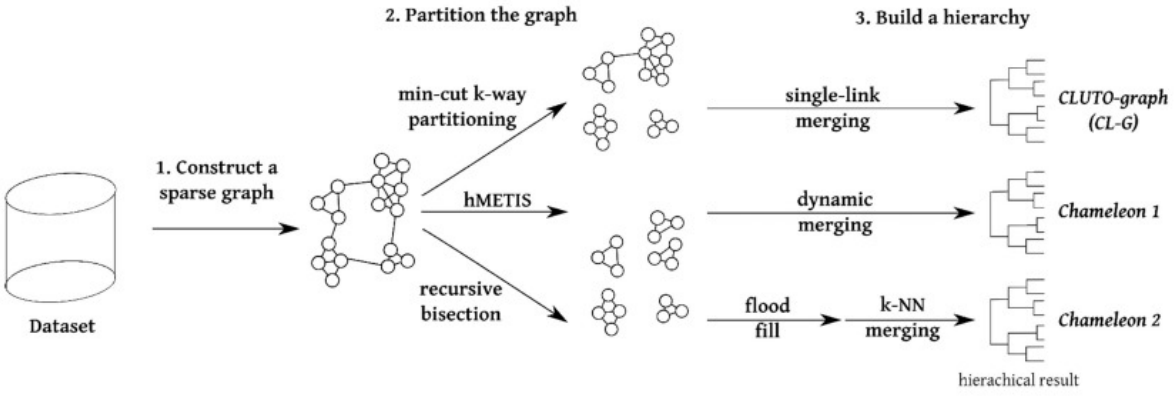

FIGURE 2.6: Chameleon Family Approaches [16]

2.3.1 Generating Symmetrical k-NN Graph

While Chameleon 1 generates asymmetric KNN graph, Chameleon 2 generates symmetrical k-NN graph, which has fewer edges than the asymmetric KNN graph. The main difference between asymmetric and symmetric KNN graph is, in asymmetric KNN graph, each node is connected to it's k nearest neighbors whereas in symmetric KNN graph, an edge connecting two nodes is present only if both are among the k nearest neighbours of each other. This reduction of the number of connections between different clusters ultimately enhances the quality of the clustering outcomes.

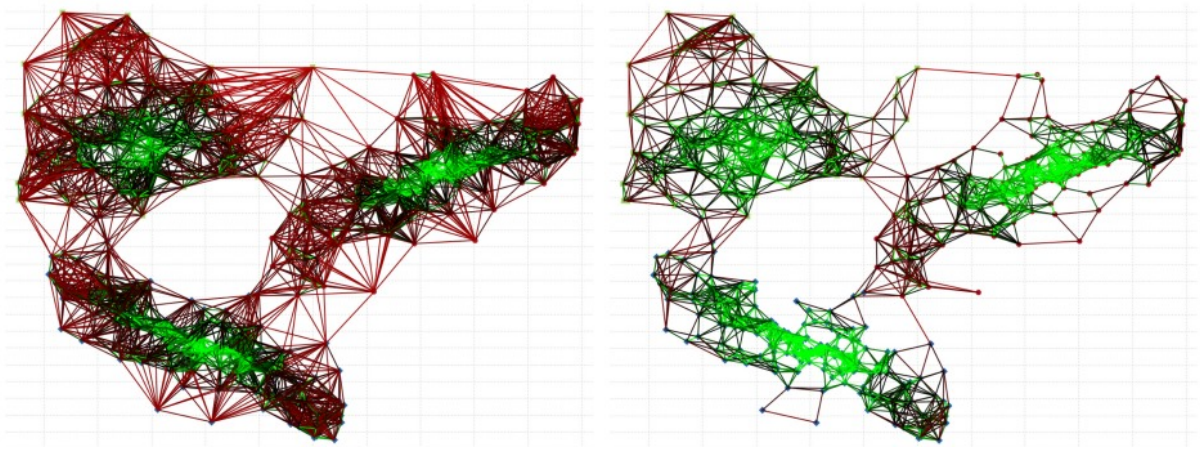

FIGURE 2.7: Asymmetric v/s Symmetric K-NN graph for 18 nearest neighbors [16]

2.3.2 Partitioning the k-nearest neighbor graph in Ch2

Chameleon 1 utilizes the hMETIS algorithm to partition the graph, but this algorithm is nondeterministic. To address the challenge of implementing further enhancements, an alternative partitioning technique is introduced, which relies on the recursive Fiduccia-Mattheyses bisection method [17]. The Fiduccia–Mattheyses bisection is repeated until each cluster contains a maximum of p_{max} objects, where p_{max} is the maximum number of data points in a single partition.

2.3.3 Partition Refinement using Flood Fill Algorithm

In their attempt to achieve a balanced partitioning with minimal edge cuts, both hMETIS and recursive bisection methods may occasionally generate clusters that encompass items that are distant and disconnected. Unfortunately, these clusters seriously affect

the outcome and are not able to be corrected during the merging phase. To tackle this problem, we apply the flood-fill method, a technique that systematically identifies connected components in the partitioned graph through recursive steps. Flood fill splits a cluster into separate, connected clusters when it consists of several disconnected components.

2.3.4 Improved Similarity Measures in Merging Phase

The most crucial enhancement introduced in Chameleon 2 involves refining the similarity measures, such as Relative Interconnectivity and Relative Closeness, compared to Chameleon 1.

First let us define some terms that we are gonna be using in the improved Similarity Measures,

$$s(C_i) = \sum_{e \epsilon C_i} w(e)$$

$$\bar{s}(C_i) = \frac{s(C_i)}{|E_{C_i}|}$$

where $s(C_i)$ is sum of weights of edges in the cluster C_i and $\bar{s}(C_i)$ is the average weight of edges in the cluster C_i.

With the removal of edge weights, the improved relative inter-connectivity measure now only considers the ratio of the number of edges,

$$R_{IC}(C_i, C_j) = \begin{cases} 1 & \text{for } |E_{C_i}| \cup |E_{C_j}| = 0 \\ \frac{|E_{C_{i,j}}|}{min(|E_{C_i}|, |E_{C_j}|)} * \rho(C_i, C_j)^{\beta} & \text{for } |E_{C_i}| \cap |E_{C_j}| > 0 \end{cases}$$

$$R_{CL}(C_i, C_j) = \begin{cases} m_{fact} * \frac{\bar{s}(C_i, C_j)}{s(C_i) + s(C_j)} & \text{for } |E_{C_i}| \cup |E_{C_j}| = 0 \\ (|E_{C_i}| + |E_{C_j}|) * \frac{\bar{s}(C_i, C_j)}{s(C_i) + s(C_j)} & \text{for } |E_{C_i}| \cap |E_{C_j}| > 0 \end{cases}$$

In this context, $|E_{C_{i,j}}|$ denotes the number of edges between clusters C_i and C_j. This part of the equation encourages the merging of clusters that have a significant number of connections compared to the connections within each cluster.

By incorporating the ρ factor, the algorithm is discouraged from merging clusters that have different densities and instead focuses on identifying clusters with similar average distances between points:

$$\rho(C_i, C_j) = \frac{min(\overline{s}(C_i), \overline{s}(C_i))}{max(\overline{s}(C_i), \overline{s}(C_i))}$$

We come across a special case for both closeness and interconnectivity: a cluster made up of just one node. In this scenario, the cluster lacks edges, resulting in zero similarity. Such clusters are prone to incorrect merging, contributing to a deterioration in the overall outcome. Despite configuring the partitioning algorithm to generate clusters with an equal number of objects, this issue still exists.Therefore, it is essential to deal with this matter in the merging stage.

We only compute the pair's R_{CL} when determining the similarity of two clusters when one of them is a singleton cluster. The final cluster similarity is then obtained by multiplying the R_{CL} by a constant m_{fact}. This strategy helps preempt potential issues in the later stages.

The adjusted relative closeness and interconnectivity combine to produce a similarity that is similar to the original Chameleon's structure. Therefore, the similarity in Chameleon 2 is equal to:

$$S_{Ch2}(C_i, C_j) = R_{CL}(C_i, C_j)^{\alpha} * R_{IC}(C_i, C_j)^{\beta}$$

This formulation of the Similarity Measure is more efficient in computation and, in the majority of instances, yields superior results compared to the original Ch1 similarity.

2.4 Approximate Nearest Neighbors

Generating the KNN graph is the most time consuming process of the Chameleon algorithm. The Exact KNN algorithm takes O(n) for a single query point, and for n points it boils down to O(n2). The approximate algorithms makes a small accuracy trade-off that could result in significant performance advantages. ANN search algorithms build a data structure called index from the dataset, and employ different algorithms to it to attain sub-linear and logarithmic time complexities. Here, we have integrated two ANN techniques with our clustering algorithm, namely FLANN and HNSW. These algortihms are explained in the next chapter.

Chapter 3

Approximate Nearest Neighbor Search

As discussed in the end of last chapter ANN algorithms can reduce the Time Complexity from $O(n^2)$ to $O(n \log n)$ (logarithmic) or $O(log^k n)$ (poly-logarithmic) T.Cs. In our work, we have integrated two popular ANN methods, namely Randomized k-d Tree implemented using FLANN library and Hierarchical Navigable Small World (HNSW) implemented using NMSlib Library. The following sections give a brief description abput these algorithms.

3.1 Hierarchical Navigable Small World

HNSW is a proximity graph based algorithm. It is a state-of-art ANN algorithm. The main philosophy behind HNSW is to construct a graph where a path between any pair of vertices could be traversed in a small number of steps. To understand HNSW we must understand to advanced data structures that contributed heavily to HNSW: the **probability skip list**, and **navigable small world graphs**.

3.1.1 Probability Skip List

A skip list is a probabilistic data structure designed to facilitate $O(\log n)$ average time complexity for the insertion and retrieval of elements in a sorted list. It comprises multiple layers of linked lists, with the bottom layer representing the original linked list containing all elements. As one ascends through the layers, the number of skipped elements grows, leading to a reduction in the number of connections.

The search begins at the highest level, comparing each element with the target value. If the value is less than or equal to the element, the algorithm advances to the next element; otherwise, it descends to a lower layer with more connections and repeats the process. This continues until the algorithm reaches the lowest layer and locates the

desired node. HNSW maintains a similar layered structure, featuring longer edges in the upper layers for swift searches and shorter edges in the lower layers to enhance precision in search operations.

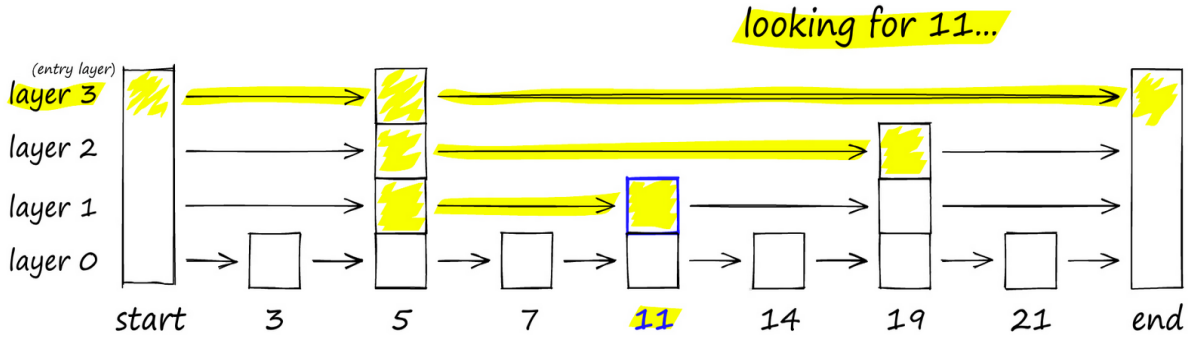

FIGURE 3.1: Example of Probability Skip List

3.1.2 Navigable Small World

A navigable small world is a graph characterized by polylogarithmic search complexity using greedy routing. Routing involves initiating the search from low-degree vertices and concluding at high-degree vertices. Due to the sparse connections of low-degree vertices, the algorithm swiftly traverses between them to efficiently reach the vicinity where the nearest neighbor is likely located. Subsequently, the algorithm progressively refines its search, transitioning to high-degree vertices to pinpoint the nearest neighbor within that specific region.

The use of greedy routing in this method doesn't assure the identification of the exact nearest neighbor, as decisions are based solely on local information at each step. An issue with the algorithm is early stopping. To resolve this issue multiple entry points can be taken and final answer could be found out by majority voting.

The NSW graph is built by shuffling dataset points and inserting them one by one in the current graph. When a new node is inserted, it is then linked by edges to the M nearest vertices to it. Typically, during the initial stages of graph construction, long-range edges are likely to be established, and these edges hold significance in facilitating graph navigation.

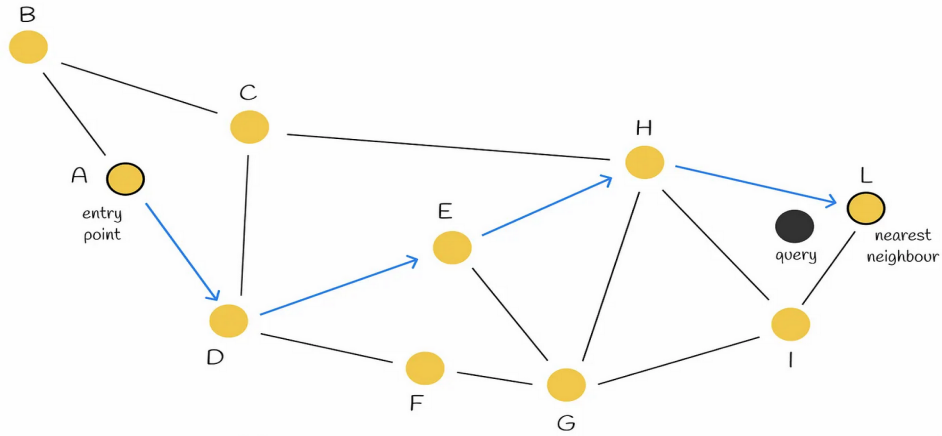

FIGURE 3.2: Searching in NSW Graph

3.1.3 Back to HNSW

HNSW represents a natural progression from NSW, drawing inspiration from Pugh's probability skip list structure and incorporating a hierarchical multi-layer approach. Adding hierarchy in NSW leads to a graph where links are distributed across various layers, with the longest links at the top layer and the shortest at the bottom layer

In the search process, we commence at the top layer, encountering the longest links, often associated with higher-degree vertices that span multiple layers. This aligns with the default zoom-in phase as described in NSW.

Similar to the NSW approach, we navigate edges in each layer using a greedy strategy, progressing towards the nearest vertex until reaching a local minimum. However, unlike NSW, at this juncture, we transition to the current vertex in a lower layer and recommence the search. This process iterates until the local minimum of our bottom layer, layer 0, is identified.

3.1.4 HNSW Graph Construction and Main Parameters

In the process of constructing the graph, vectors are inserted iteratively, one by one. The parameter **L** represents the number of layers in the graph. Every node is randomly assigned an integer **l** indicating the maximum layer at which this node can present in the graph. For example, if **l = 1**, then the node can only be found on layers 0 and 1. The authors select **l** randomly for each node with an exponentially decaying probability

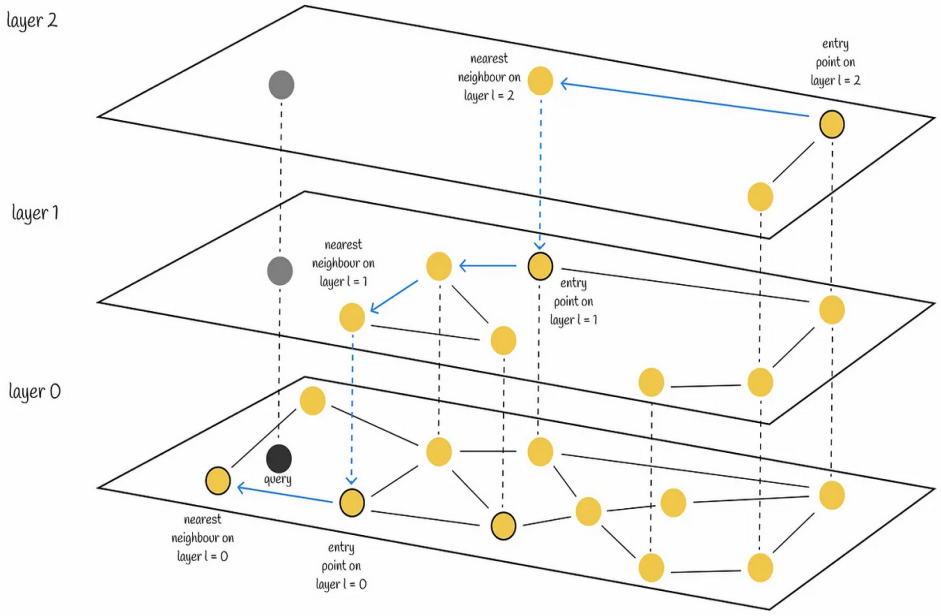

FIGURE 3.3: Searching in NSW Graph

distribution normalized by the non-zero multiplier **mL** (mL = 0 results in a single layer in HNSW and non-optimized search complexity). Normally, the majority of l values should be equal to 0, so most of the nodes are present only on the lowest level. The larger values of **mL** increase the probability of a node appearing on higher layers.

Reducing **mL** is a method to diminish overlap, but it's crucial to consider that decreasing **mL** also, on average, results in more traversals during a greedy search on each layer. Therefore, it's essential to find a balance between minimizing overlap and managing the number of traversals. The authors suggest selecting an optimal **mL** value, specifically **1 / ln(M)**, to strike a balance between these two factors, where parameter **M** is the number of neighbors we add to each vertex on insertion.

After a vertex is assigned the value l, there are two phases of its insertion, they are:

- The algorithm initiates its process from the upper layer, greedily identifying the nearest node. This discovered node serves as the entry point to the next layer, perpetuating the search process. Upon reaching layer l, the insertion then advances to the second step
- Commencing from layer l, the algorithm inserts the new node at the current layer. Subsequently, it follows a procedure similar to step 1, but instead of seeking only

one nearest neighbor, it performs a greedy search for efConstruction (a hyperparameter) nearest neighbors. From this set, M out of efConstruction neighbors are selected, and edges are established from the inserted node to these chosen neighbors. Following this, the algorithm descends to the next layer, and each of the identified efConstruction nodes serves as an entry point. The algorithm concludes after the new node and its corresponding edges are inserted on the lowest layer, i.e. the layer 0.

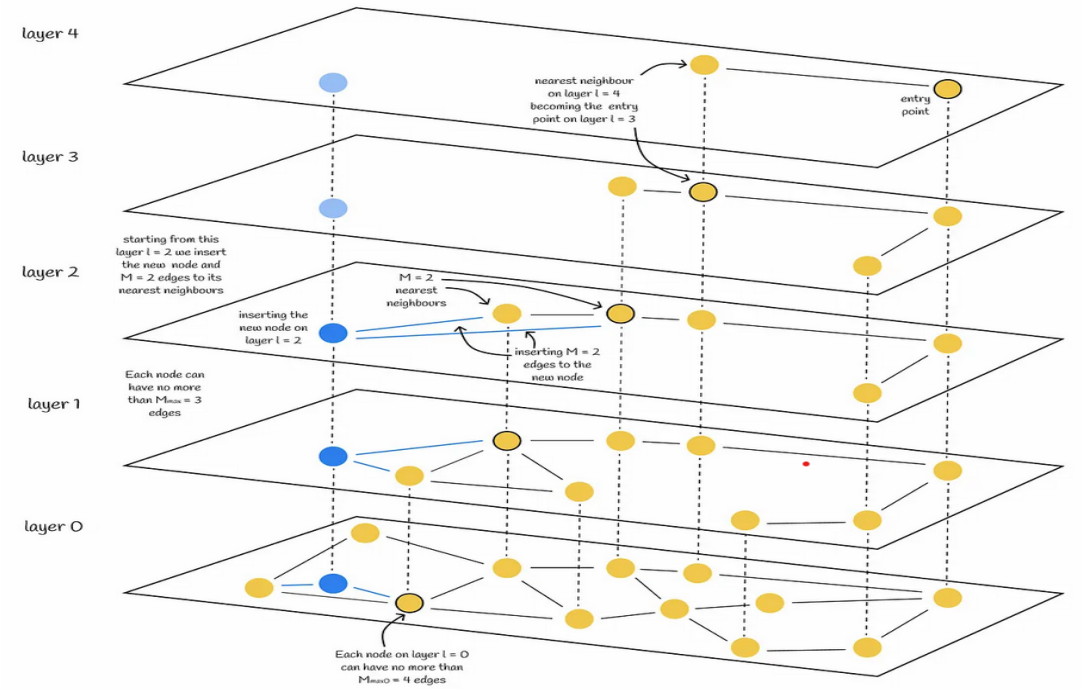

FIGURE 3.4: Searching in NSW Graph

The authors of the HNSW provided valuable perspectives on the selection of hyperparameters. following are their insights:

- As per several experiments, optimal M values typically range from 5 to 48. Smaller M values are preferable for lower recalls or low-dimensional data, whereas larger M values are more suitable for high recalls or high-dimensional data.
- Greater values of efConstruction indicate a more extensive search, involving the exploration of more candidates. However, this comes at the cost of increased computational demands. The authors advise selecting an efConstruction value that yields a recall close to 0.95–1 during training.

- Furthermore, there is another crucial parameter, M_{max}, representing the maximum number of edges a vertex can have. It is advisable to select a value for M_{max} close to 2 * M. Values exceeding 2 * M may result in performance degradation and excessive memory usage. Conversely, setting M_{max} equal to M leads to suboptimal performance at high recall.

3.2 Randomized K-d Tree

The FLANN library provides various ANN methods, three to be precise, they are **Hierarchical k-means**, **Randomized k-d tree** and **Standard Linear Search**. In our work, we incorporated the second one because of two main reasons: it gives higher accuracy compared to its counterparts, and the second one is it is easier to fine-tune. Also, for lower dimensions, it has been shown that the time complexity of the k-d trees method for finding nearest neighbors for n points out to be $O(n \log n)$.To understand the Randomized k-d tree we must first understand the k-d tree.

3.2.1 Standard k-d Tree and Randomized k-d Tree

A K-D Tree, also known as a K-Dimensional Tree, functions as a binary search tree in which each node contains data representing a point in K-Dimensional space. Essentially, it serves as a space-partitioning data structure designed to organize points within a K-Dimensional space. In this tree structure, a non-leaf node partitions the space into two distinct halves, referred to as half-spaces, i.e. a non-leaf node acts as a hyperplane. Points located to the left of this partition are managed by the left subtree of the node, while those to the right of the partition are handled by the right subtree.

Following is the high-level overview of the process of constructing a k-d tree:

- **Choose a dimension**: Select the dimension along which to split the points. This is often done by cycling through the dimensions in order. For example, if you're working in 2D space, you might alternate between splitting along the x-axis and the y-axis.
- **Choose a pivot**: Identify a pivot point in the selected dimension. This can be done in various ways, such as selecting the midpoint of the range of values along the chosen dimension.

- **Partition the points**: Divide the points into two sets - those that are on one side of the pivot and those on the other side. This forms a binary partition along the chosen dimension.
- **Create nodes**: Create a node in the tree representing the pivot point and the dimension along which the split was made. Recursively repeat the process for the two sets of points on either side of the pivot until each point is in its own leaf node.

The key difference between a standard and a randomized k-d tree is that in a Randomized k-d tree, the perpendicular hyperplane is centered at the mean of the splitting dimension values of all data points. Also, the splitting dimension is chosen at random from the top-5 dimensions that have the largest sample variance. Multiple independent randomized k-d trees are generated as an index to improve the accuracy of the algorithm. Interestingly, while using this algorithm, the number of trees to be built, i.e **numTrees** is the only parameter that needs to be passed in the library method.

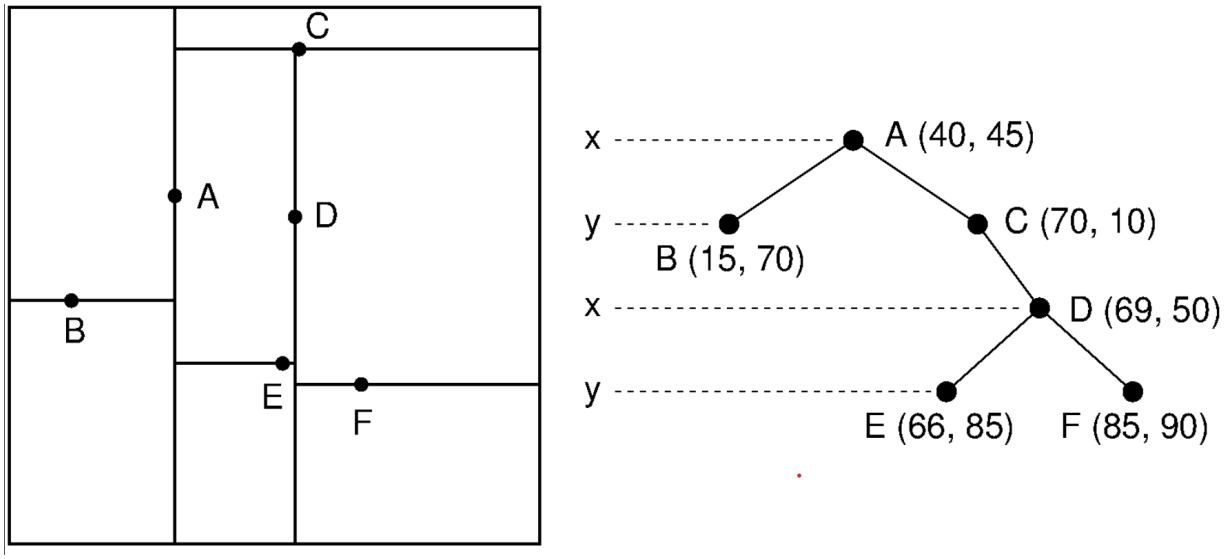

FIGURE 3.5: Example of k-d (2-d) tree

3.2.2 Nearest Neighbour Search in k-d tree

Commencing from the root node, the process navigates down the tree recursively, choosing the left or right direction based on whether the **query point** is smaller or larger than the current node in the split dimension. While traversing the tree, the algorithm identifies and records the node with the shortest distance to the target point as the "current

best." Upon reaching a leaf node, the recursion unwinds, and at each node, the following steps are executed:

- If the current node is closer than the current best, it becomes the new current best.
- The algorithm assesses whether there might be points on the opposite side of the splitting plane that are closer to the target point than the current best. This involves intersecting the splitting hyperplane with a hypersphere around the target point, with the sphere's radius matching the current nearest distance. A straightforward comparison is used to determine whether the difference between the splitting coordinate of the target point and the current node is less than the distance from the target point to the current best.
- If the hypersphere intersects the plane, indicating the possibility of closer points on the other side, the algorithm descends the tree's alternate branch from the current node, continuing the recursive search.
- If the hypersphere does not intersect the splitting plane, the algorithm ascends the tree, eliminating the entire branch on the opposite side of that node. The search concludes when the algorithm completes this procedure for the root node.

The following example provides a pictorial walkthrough of the above process:

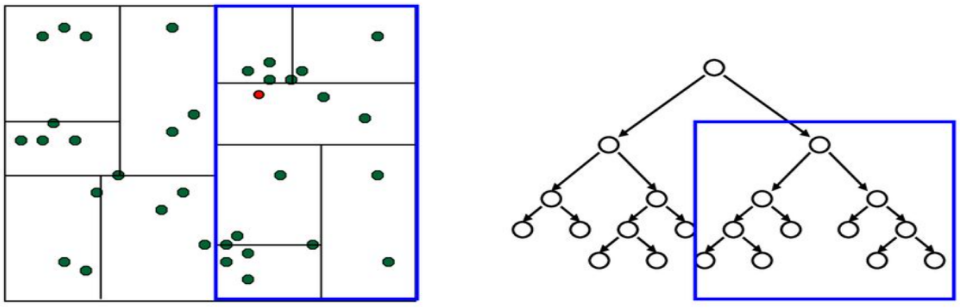

FIGURE 3.6: Nearest Neighbour Search in k-d tree, Step-1

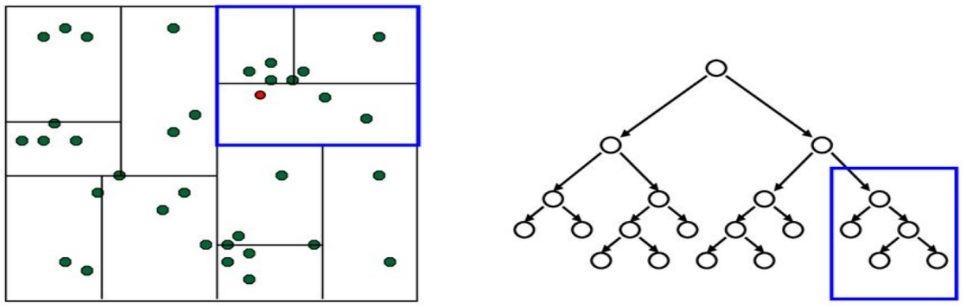

FIGURE 3.7: Nearest Neighbour Search in k-d tree, Step-2

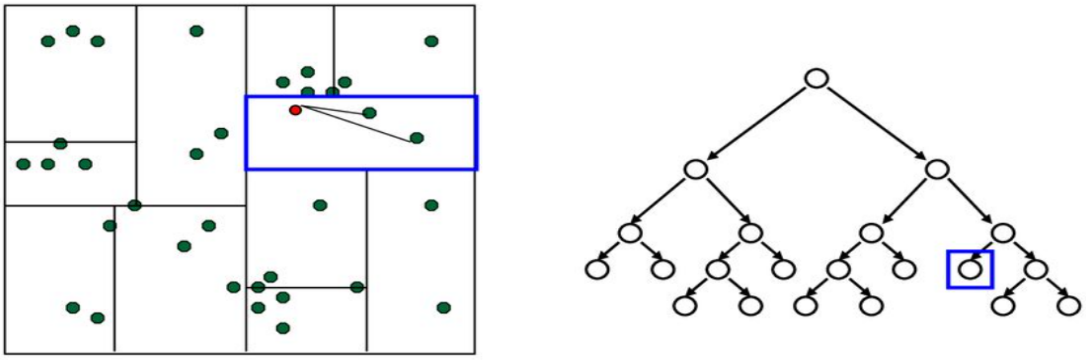

FIGURE 3.8: Nearest Neighbour Search in k-d tree, Step-3

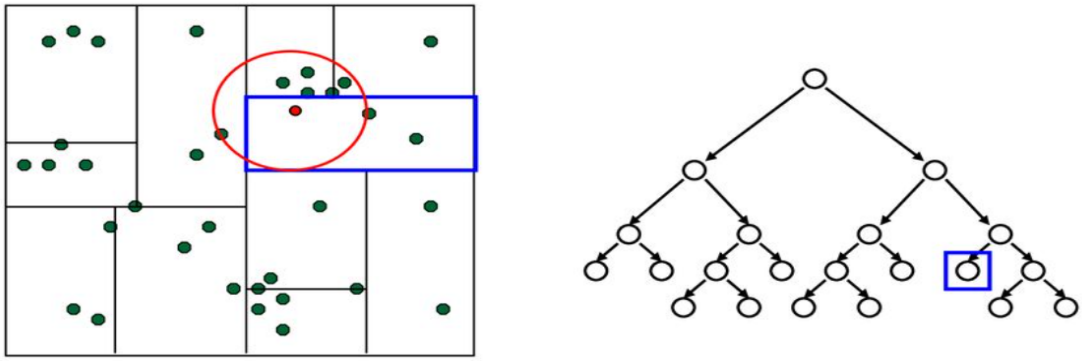

FIGURE 3.9: Nearest Neighbour Search in k-d tree, Step-4

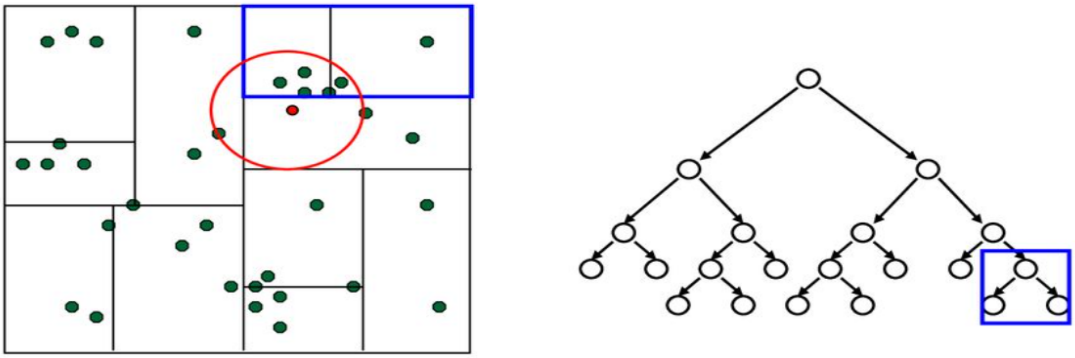

FIGURE 3.10: Nearest Neighbour Search in k-d tree, Step-5

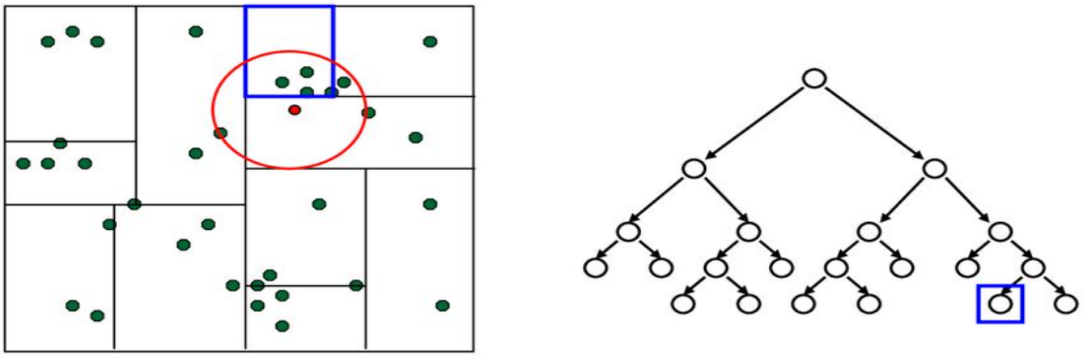

FIGURE 3.11: Nearest Neighbour Search in k-d tree, Step-6

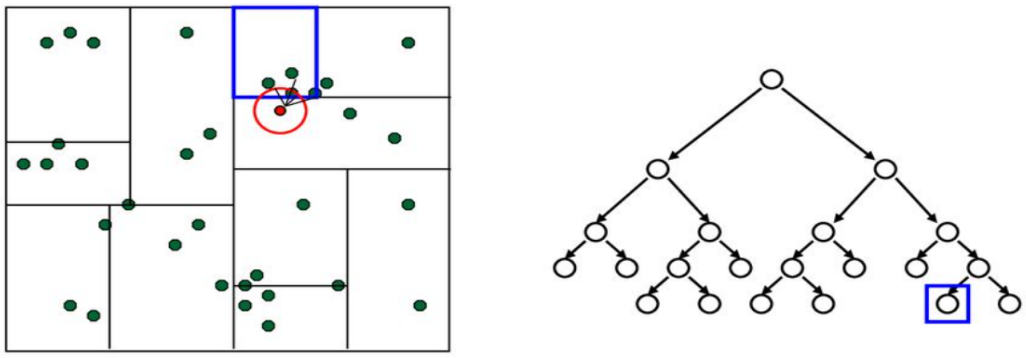

FIGURE 3.12: Nearest Neighbour Search in k-d tree, Step-7

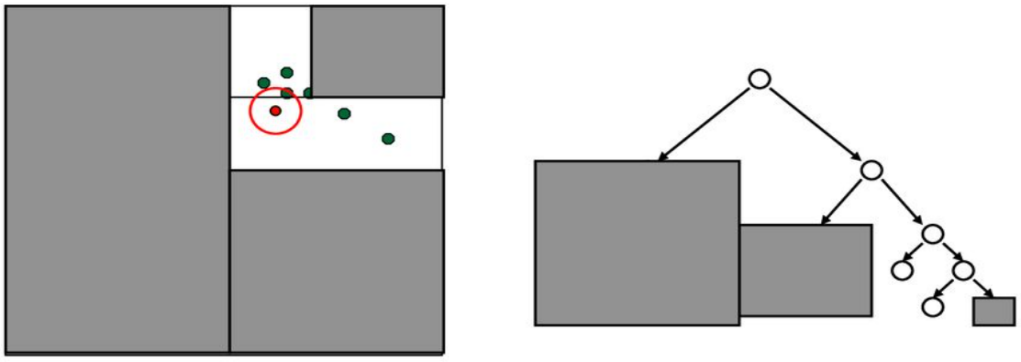

FIGURE 3.13: Nearest Neighbour Search in k-d tree, Step-8

Chapter 4

Experiments and Results

In this chapter, we will go through various datasets we have used for our project, and the results obtained. We compared our approximate nearest neighbour search-based modified Chameleon 2 algorithm to a number of well-known clustering algorithms. Section 4.1 gives an overall view of the datasets. Section 4.2 discusses about the metrics used for evaluation. Section 4.3, then discusses about the results obtained.

4.1 Datasets

Our benchmark's section purposefully contains both 2D datasets and 3D datasets. featuring easily recognizable structures, detectable by humans. Furthermore, some datasets are intentionally contaminated with noisy clusters in order to evaluate the robustness of the algorithm.

We have used a total of 32 datasets to evaluate our algorithm. A set of six datasets, namely *twodiamonds, target, chainlink, atom, lsun, wingnut* were intentionally crafted to pose challenges that cannot be fully addressed by conventional clustering algorithms.

Each dataset consist of around 200 to 10000 datapoints, number of classes ranging from 2 to 31. Only 8 of the 32 datasets (impossible, zelink4, dpb, cure-t2-4k and the 4 cluto datasets) contain nosiy clusters and remaining does not contain any noise.

Assessing the effectiveness of clustering algorithms poses challenges, and various methods have been employed in related research. Although unsupervised criteria, as demonstrated by [18], can be applied during clustering, their efficacy relies on the features of the data. The most common technique involves evaluating clustering results against external labels, commonly referred to as ground truth.

Below are information (Table 4.1) and visualisations of some of the datasets,

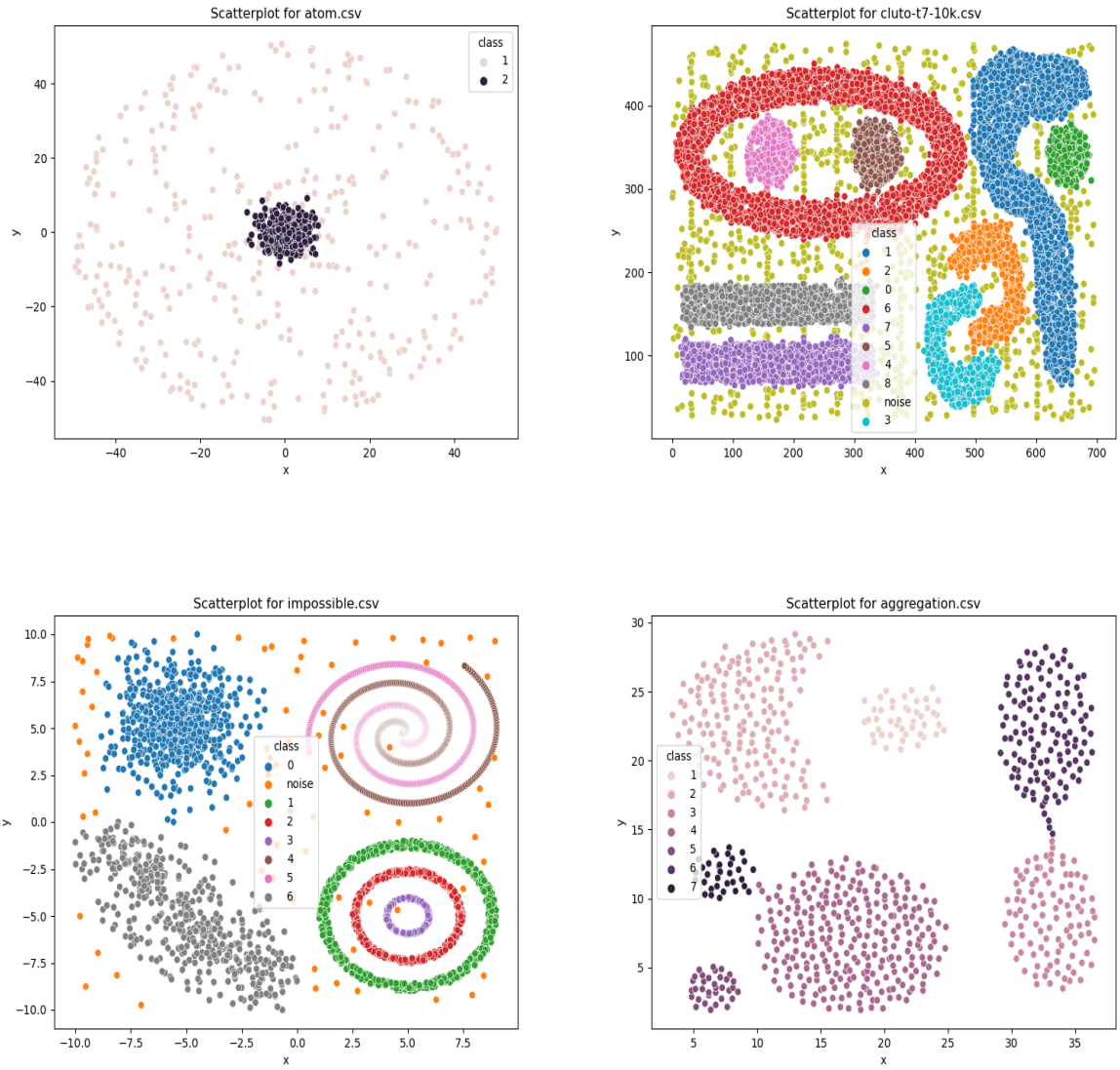

FIGURE 4.1: Visualisation of Datasets

4.2 Evaluation Metrics

We have used NMI(Normalised Mutual Information) Score and Rand Score as the evaluation metrics for the datasets.

NMI Score : NMI gauges the shared information between the actual class labels and the forecasted clusters, adjusted based on the entropy of the real and anticipated labels. It provides a value between 0 and 1, where higher values indicate better clustering performance. NMI is advantageous because it considers both the completeness and

Dataset	Dimension	Data Points	Noise size (%)	Classes
3-spiral	2	312	-	3
aggregation	2	788	-	7
atom	3	800	-	2
chainlink	3	1,000	-	2
cluto-t4.8k	2	8,000	764 (9.55%)	7
cluto-t5.8k	2	8,000	1153 (14.41%)	9
cluto-t7.10k	2	10,000	792 (9.92%)	8
cluto-t8.8k	2	8,000	323 (4.04%)	8
compound	2	399	-	6
cure-t2-4k	2	4,000	200 (4.76%)	7
D31	2	3,100	-	31
dense-disk-5k	2	5,000	-	2
DS-850	2	850	-	5
diamond9	2	3000	-	9
disk-in-disk	2	4,600	-	2
dpb	2	4,000	657 (16.43%)	6
flame	2	240	-	2
impossible	2	3,673	78 (2.12%)	8
jain	2	373	-	2
long1	2	1,000	-	2
longsquare	2	900	-	6
lsun	2	400	-	3
pathbased	2	300	-	3
s-set1	2	5,000	-	15
sizes1	2	1,000	-	4
smile1	2	1,000	-	4
spiralsquare	2	1,500	-	6
target	2	770	-	6
twodiamonds	2	800	-	2
wingnut	2	1,016	-	2
zelnik4	2	622	138 (22.19%)	5

TABLE 4.1: Dataset Information

homogeneity of the clustering, providing a balanced assessment.

Rand Score : Rand Score gauges how well the predicted clusters align with the true clusters by counting the pairs of samples that are correctly or incorrectly grouped. Higher values of the Rand Score, which ranges from 0 to 1, denote better clustering agreement. It is divided into two components: the Rand Index and the Adjusted Rand Index, with the latter compensating for the effects of chance.

In summary, both NMI and Rand Score offer valuable insights into the accuracy and agreement of clustering results, considering different aspects of the clustering quality. Researchers and practitioners often use a combination of these metrics to comprehensively evaluate the performance of clustering algorithms.

4.3 Results

Various enhancements have been implemented in the Ch2 Algorithm with the integration of nearest neighbor search. The column labeled Ch2 + FLANN represents the results when FLANN is employed for ANN search, and the same applies to NMSLIB. Table 4.2 contains a list of these improvements specifics.

Table 4.2 presents Normalized Mutual Information (NMI) values for different algorithms, specifically the Ch2 algorithm with FLANN, Ch2 with NMSLIB, and the standard Ch2 algorithm.

Ch2 + FLANN and Ch2 + NMSLIB consistently achieve NMI values close to or equal to 1 in several datasets, such as 3-spiral, chainlink, jain, long1, lsun, target, triangle1, twodiamonds, and wingnut.

On average, both Ch2 + FLANN and Ch2 + NMSLIB outperform the standard Ch2 algorithm, with average NMI values of 0.898, 0.929, and 0.964, respectively. This suggests that incorporating these search methods tends to enhance the clustering quality across diverse datasets. The standard deviation values (0.114, 0.086, 0.049) indicate the

Dataset	**Ch2 + FLANN**	**Ch2 + NMSLIB**	**Ch2**
3-spiral	**1**	**1**	**1**
aggregation	0.996	0.993	0.992
atom	0.994	0.991	**1**
chainlink	**1**	**1**	**1**
cluto-t4.8k	0.825	0.821	0.893
cluto-t5.8k	0.812	0.822	0.864
cluto-t7.10k	0.652	0.838	0.912
cluto-t8.8k	0.804	0.812	0.944
compound	0.927	0.955	0.992
cure-t2-4k	0.817	0.912	0.967
D31	0.921	0.922	0.957
dense-disk-5k	0.67	0.779	0.908
diamond9	0.996	0.991	0.993
disk-in-disk	0.631	0.873	0.99
dpb	0.692	0.649	0.81
DS-850	0.963	0.979	0.984
flame	0.935	0.927	0.927
impossible	0.872	0.922	0.969
jain	**1**	**1**	**1**
long1	**1**	**1**	**1**
longsquare	0.916	0.993	0.981
lsun	**1**	**1**	**1**
pathbased	0.919	0.905	0.887
s-set1	0.95	0.9	0.997
sizes1	0.911	0.915	0.909
smile1	0.967	**1**	**1**
spiralsquare	0.953	0.998	0.987
target	**1**	**1**	**1**
triangle1	**1**	**1**	**1**
twodiamonds	**1**	**1**	**1**
wingnut	**1**	**1**	**1**
zelnik4	0.827	0.845	0.987
Average	0.898	0.929	0.964
Std. Deviation	0.114	0.086	0.049

TABLE 4.2: NMI Values for various Algorithms

variability in performance across datasets. Lower standard deviations generally suggest more consistent performance.

In summary, the table indicates that incorporating FLANN or NMSLIB for nearest neighbor search tends to improve clustering performance, with some datasets benefiting more than others. The average NMI values show a positive impact on clustering quality, emphasizing the effectiveness of these enhancements in diverse scenarios.

Table 4.3 illustrates the variation in NMI values across all 32 datasets, comparing the percentage differences between Ch2 + FLANN, Ch2 + NMSLIB, and the standard Ch2 clustering methods.

In the case of smaller datasets, including *3-spiral, atom, jain, long1, lsun, target, triangle, twodiamonds, and wingnut*, the NMI values remain consistent for both the Ch2 + FLANN and Ch2 + NMSLIB methods when compared to the standard Ch2 algorithm. The clustering performance on these datasets appears to be relatively stable, showing little to no deviation with the integration of FLANN or NMSLIB for approximate nearest neighbor search.

There is a significant decline in performance for Ch2 + FLANN, particularly evident in the 'disk-in-disk' dataset, where a huge decrease of 36.3% is observed. Similarly, Ch2 + NMSLIB exhibits a considerable deviation from the standard Ch2 algorithm, notably on the 'dpb' dataset. These considerable declines can be attributed to the inherent challenge presented by these datasets, where clusters lack clear separation, making the clustering task more intricate. The distinctive characteristics of the 'disk-in-disk' and 'dpb' datasets, marked by less well-defined cluster boundaries, contribute to the observed substantial decreases in NMI values for Ch2 + FLANN and Ch2 + NMSLIB.

4.4 Parameters

Both approximate nearest neighbor search algorithms, FLANN and HNSW, involve the utilization of multiple parameters. Subsection 4.4.1 discusses the parameters used in FLANN and Subsection 4.4.2 discusses the parameters used in HNSW. They are the parameters which are used to get the highest possible NMI values for the corresponding datasets.

Dataset	FLANN - Ch2(%)	NMSLIB - Ch2(%)
3-spiral	0	0
aggregation	0.37	0.07
atom	-0.6	-0.9
chainlink	0	0
cluto-t4.8k	-7.61	-8.06
cluto-t5.8k	-6.02	-4.86
cluto-t7.10k	-28.51	-8.11
cluto-t8.8k	-14.83	-13.98
compound	-6.59	-3.73
cure-t2-4k	-15.51	-5.69
D31	-3.8	-3.67
dense-disk-5k	-26.25	-14.21
diamond9	0.26	-0.19
disk-in-disk	-36.3	-11.82
dpb	-14.57	-19.88
DS-850	-2.13	-0.48
flame	0.91	-0.01
impossible	-9.99	-4.89
jain	0	0
long1	0	0
longsquare	-6.62	1.18
lsun	0	0
pathbased	3.59	2.04
s-set1	-4.71	-9.73
sizes1	0.26	0.63
smile1	-3.28	0
spiralsquare	-3.49	1.07
target	0	0
triangle1	0	0
twodiamonds	0	0
wingnut	0	0
zelnik4	-16.21	-14.39

TABLE 4.3: Percentage Change in NMI Values for FLANN and NMSLIB Compared to Ch2

4.4.1 FLANN Parameters

There are multiple parameters used in FLANN like the number of nearest neighbors to each datapoint, the type of algorithm used, number of trees used for the generation of forest and the merging phase parameters α and β. These are described as follows:

- **Number of Nearest Neighbors :** This parameter determines the number of closest neighbours that will be considered for each data point in the dataset. The Euclidean norm is applied for computing the distance between two datapoints.
- **Algorithm :** Specifies the algorithm used for nearest neighbor search, such as kd-tree, kmeans, linear, hierarchical clustering, etc.
- **Number of Trees :** The Number of Trees parameter determines how many randomized kd-trees are created in the forest. Each tree serves as an independent structure for organizing and partitioning the data points in a way that facilitates efficient nearest neighbor searches. The use of multiple trees introduces an element of randomness and diversity in the search process.
- **Merging phase parameters α and β :** The parameters α and β are user-defined variables that determine the balance between either the degree of proximity or the degree of interconnecteivity usesd in the similarity measure between two clusters while merging the clusters.

These parameters offer adaptability when setting up FLANN according to the data's characteristics and the preferred trade-offs between speed and accuracy in the search for nearest neighbors. Modifying these parameters enables users to customize FLANN's performance to meet specific application needs. These parameters for all the 32 datasets are specified in the Table 4.4.

4.4.2 HNSW Parameters

Before delving into the details of the HNSW algorithm's parameters, it's essential to understand that these parameters significantly influence the behaviour and efficiency of the algorithm. Each parameter serves a specific purpose, offering users the flexibility to adapt the algorithm to different datasets and use cases. Whether it's controlling the graph's density, influencing the exploration during search, or determining the storage strategy, these parameters allow users to strike a balance between factors like search

Dataset	**Neighbors**	**Trees**	**Algorithm**	α	β
3-spiral	4	4	kdtree	2	1
aggregation	10	4	kdtree	2	1
atom	10	8	kdtree	2	1
chainlink	10	8	kdtree	2	1
cluto-t4-8k	12	8	kdtree	4	2
cluto-t5-8k	11	8	kdtree	4	2
cluto-t7-10k	14	12	kdtree	2	3
cluto-t8-8k	20	12	kdtree	2	3
compound	10	8	kdtree	2	1
cure-t2-4k	19	12	kdtree	2	1
D31	10	4	kdtree	2	1
dense-disk-5000	13	12	kdtree	1	4
diamond9	10	8	kdtree	2	1
disk-in-disk	6	8	kdtree	2	4
dpb.csv	9	12	kdtree	3	4
DS-850	12	4	kdtree	2	1
flame	12	8	kdtree	2	1
impossible	12	6	kdtree	2	1
jain	6	4	kdtree	2	1
long1	11	12	kdtree	2	1
longsquare	6	4	kdtree	2	1
lsun	6	4	kdtree	2	1
pathbased	9	8	kdtree	2	1
s-set1	21	12	kdtree	2	1
sizes1	11	8	kdtree	2	1
smile1	9	12	kdtree	2	1
spiralsquare	11	8	kdtree	2	1
target	9	4	kdtree	2	1
triangle1	7	8	kdtree	2	1
twodiamonds	7	8	kdtree	2	1
wingnut	5	12	kdtree	2	1
zelink	9	12	kdtree	2	1

TABLE 4.4: FLANN Parameters

accuracy, query time, and memory consumption. Careful selection and fine-tuning of these parameters play a key role in enhancing the algorithm's efficiency and effectiveness in tasks involving approximate nearest neighbor searches.

- **Number of Nearest Neighbors :** Refer subsection 4.4.1 to understand more about these parameters.
- **Maximum Neighbors (M) :** This parameter defines the maximum number of neighbors that can be connected to a single node in the graph. A larger value of **M** results in a denser graph and more exhaustive search, but it also increases memory requirements and search time.
- **Exploration Factor during Construction (efC) :** This parameter controls the number of neighbors considered during the construction of the graph. A higher value increases the quality of the constructed graph but also requires more computational resources.
- **Number of Threads :** This parameter specifies the number of threads used by the algorithm during the construction phase. It controls parallelization during construction, potentially improving efficiency on multi-core systems.
- **Merging phase parameters α and β :** Refer subsection 4.4.1 to understand more about these parameters.

These parameters collectively offer users the ability to fine-tune the HNSW algorithm to match the characteristics of their data and meet specific performance requirements, considering trade-offs between factors such as search accuracy, query time, and memory consumption.

4.5 Graphical Comparision

In this section, we will explore a visual comparison of NMI values and the runtime for the Ch2 + FLANN and Ch2 + HNSW algorithms in comparison to the standard Ch2 algorithm across all 32 datasets.

4.5.1 Graphical Visualisation of NMI Values

In the following visual representation, we present a graphical comparison of Normalized Mutual Information (NMI) values among the Ch2 + FLANN, Ch2 + NMSLIB, and

Dataset	**Neighbors**	**M**	**efC**	**Threads**	α	β
3-spiral	4	10	100	4	2	1
aggregation	6	10	100	4	2	1
atom	10	10	100	4	2	1
chainlink	10	10	100	4	2	1
cluto-t4-8k	39	10	100	4	5	3
cluto-t5-8k	39	10	100	4	6	3
cluto-t7-10k	12	10	100	4	2	3
cluto-t8-8k	39	10	100	4	2	3
compound	8	10	100	4	2	1
cure-t2-4k	7	10	100	4	2	1
D31	10	10	100	4	2	1
dense-disk-5000	13	10	100	4	1	4
diamond9	12	10	100	4	2	1
disk-in-disk	20	10	100	4	2	4
dpb	35	10	100	4	3	4
DS-850	14	10	100	4	2	1
flame	12	10	100	4	2	1
impossible	9	10	100	4	2	1
jain	6	10	100	4	2	1
long1	9	10	100	4	2	1
longsquare	5	10	100	4	2	1
lsun	6	10	100	4	2	1
pathbased	7	10	100	4	2	1
s-set1	19	10	100	4	2	1
sizes1	6	10	100	4	2	1
smile1	10	10	100	4	2	1
spiralsquare	14	10	100	4	2	1
target	9	10	100	4	2	1
triangle1	7	10	100	4	2	1
twodiamonds	6	10	100	4	2	1
wingnut	5	10	100	4	2	1
zelnik4	6	10	100	4	2	1

TABLE 4.5: HNSW Parameters

the standard Ch2 algorithm across all 32 datasets. The NMI metric provides insights into the quality of clustering, and these graphs aim to illustrate how the performance of Ch2 is influenced by the incorporation of FLANN and NMSLIB for approximate nearest neighbor search.

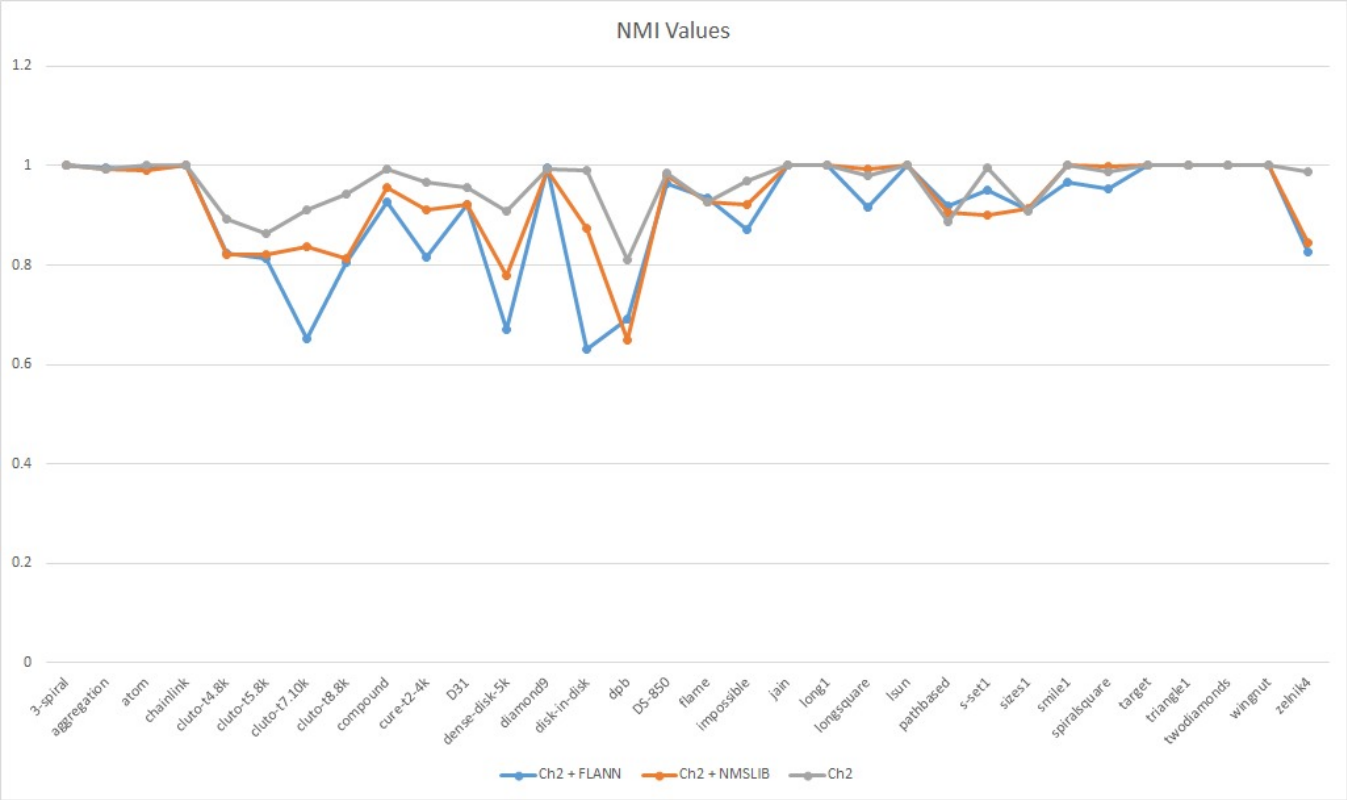

FIGURE 4.2: Graphical Visualisation of NMI Values

4.5.2 Graphical Visualisation of Run Times

In the below visual presentation, we showcase a graphical comparison of the runtime performance for the Ch2 + FLANN, Ch2 + NMSLIB, and the standard Ch2 algorithm across all 32 datasets. The runtime analysis is crucial for evaluating the computational efficiency of these algorithms. This graphical representation aims to provide a clear perspective on how the incorporation of FLANN and NMSLIB influences the runtime of the Ch2 algorithm.

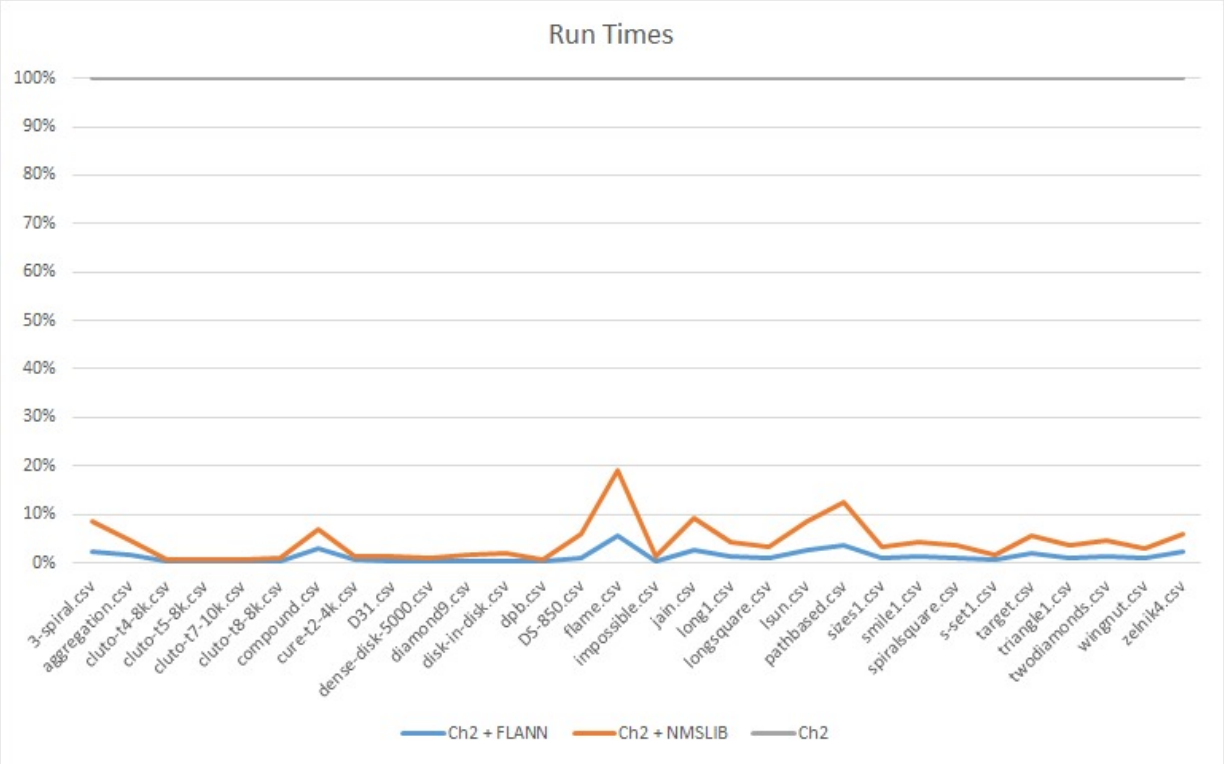

FIGURE 4.3: Graphical Visualisation of Run Times

As depicted in Figure 4.3, it becomes clear that the Ch2 + FLANN algorithm remarkably outperforms the standard Ch2 Algorithm in terms of runtime across all datasets. The Ch2 + FLANN algorithm exhibits a substantial improvement by taking only 0-10% of the time taken by the standard Ch2 Algorithm. This noteworthy enhancement underscores the efficiency gains in terms of runtime achieved through the incorporation of FLANN for approximate nearest neighbor search. Similarly, the Ch2 + NMSLIB algorithm demonstrates remarkable efficiency, consistently taking only 0-10% of the time taken by the standard Ch2 Algorithm across a diverse range of datasets. This underscores the effectiveness of leveraging NMSLIB for approximate nearest neighbor search, contributing to significant reductions in computational time. The graphical representation provides a clear visualization of these substantial improvements in runtime performance.

4.6 Two-Tailed Paired T-test

The two-tailed paired t-test is employed to statistically assess the significance of differences in Normalized Mutual Information (NMI) values between clustering results obtained using the Ch2 + FLANN and Ch2 + NMSLIB algorithms compared to the

standard Ch2 algorithm. In this analysis, the NMI values from the three algorithms are treated as paired observations, as they are obtained from the same datasets. The null hypothesis posits that there is no notable difference in the NMI values between the Ch2 + FLANN or Ch2 + NMSLIB algorithms and the standard Ch2 algorithm. Conversely, the alternative hypothesis, posits that there is a statistically significant difference. The two-tailed test is appropriate here as we are interested in detecting any deviation, whether in an increase or decrease, in NMI values.

The test yields a p-value, and if the p-value is less than a predefined significance level (typically 0.05), we deny the null hypothesis, suggesting a statistically significant disparity in clustering performance among the algorithms being compared. . This statistical analysis aids in rigorously evaluating whether the incorporation of FLANN or NMSLIB for approximate nearest neighbor search has a substantial influence on the quality of clustering, as measured by NMI, compared to the standard Ch2 algorithm.

The p-value for the comparison between Ch2 + FLANN and the standard Ch2 algorithm is found out to be **0.000818**, and similarly, the p-value for the comparison between Ch2 + NMSLIB and the standard Ch2 algorithm is calculated as **0.000787**. These p-values, both below the commonly used significance level of **0.05**, indicate a statistically significant difference in the NMI values between the algorithms being compared. In other words, there is substantial evidence to reject the null hypothesis, affirming that the clustering performance, as measured by NMI, is significantly different when employing Ch2 + FLANN or Ch2 + NMSLIB in comparison to the standard Ch2 algorithm.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this report, we've presented the Chameleon and Chameleon 2 Clustering algorithms which uses dynamic model framework to merge the clusters.In order to address the limitations of the conventional Chameleon Algorithm, we've introduced the Chameleon 2 Algorithm which generates symmetric K-NN graph instead of asymmetric K-NN graph, uses flood fill algorithm to partition the refinement and have improved similarity measures.

Approximate nearest neighbor algorithms like FLANN and HNSW are used so that we can make a small trade off of accuracy for the sake of significant performance advantages. To make sure the algorithm is strong and reliable, we tested it thoroughly using 32 different sets of data. There is a huge performance gain in terms of runtime and the NMI values are close to Ch2 and for some datasets the NMI values are lagging a little bit behind which is expected to happen because we are using approximate nearest neighbors.

FLANN is a space partitioning based method and it divides the data space into regions and organize data points in structures like trees (e.g: KD-Trees, RP-Trees, Ball Trees, etc).The objective is to selectively remove extensive portions of the space during the process of searching for neighbouring elements.

- It is suitable for medium dimensional data.
- FLANN can provide a good balance between accuracy and speed.
- But it suffers from curse of dimensionality. As the number of dimension increase, the effectiveness decreases.

HNSW is a method which is based on graph structures and uses the concept of navigable small world and probability skip list.The objective is to efficiently traverse this graph in order to identify the closest neighbors.

- It is used when maintaining the quality of one's neighbors is critical.
- It often provides very high-quality neighbors.
- But the graph construction phase can bee time taking.

5.2 Future Work

A novel approach can be developed by incorporating techniques from modern-day graph theory. This involves exploring and applying contemporary graph theory methods to formulate an innovative and effective solution. There are few optimizations that can be done in the merging phase as well, like tweaking the values of α and β. The improved algorithm can be applied to real-world datasets, including astronomical data. This involves testing and validating the algorithm's performance and effectiveness on practical and complex datasets, such as those encountered in astronomical research.

Bibliography

[1] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[2] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[3] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. "ROCK: A robust clustering algorithm for categorical attributes". In: *Information systems* 25.5 (2000), pp. 345–366.

[4] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. "CURE: An efficient clustering algorithm for large databases". In: *ACM Sigmod record* 27.2 (1998), pp. 73–84.

[5] Godfrey N Lance and William Thomas Williams. "A general theory of classificatory sorting strategies: 1. Hierarchical systems". In: *The computer journal* 9.4 (1967), pp. 373–380.

[6] Ying Zhao and George Karypis. "Criterion functions for document clustering: Experiments and analysis". In: (2001).

[7] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the royal statistical society: series B (methodological)* 39.1 (1977), pp. 1–22.

[8] Geoffrey J McLachlan and Kaye E Basford. *Mixture models: Inference and applications to clustering*. Vol. 38. M. Dekker New York, 1988.

[9] George Karypis, Euihong Han, and Vipin Kumar. "A hierarchical clustering algorithm using dynamic modeling". In: (1999).

[10] George Karypis and Vipin Kumar. "Multilevel k-way hypergraph partitioning". In: *Proceedings of the 36th annual ACM/IEEE design automation conference*. 1999, pp. 343–348.

[11] George Karypis et al. "Multilevel hypergraph partitioning: Application in VLSI domain". In: *Proceedings of the 34th annual Design Automation Conference*. 1997, pp. 526–529.

[12] George Karypis. "hMETIS 1.5: A hypergraph partitioning package". In: *http://www.cs. umn. edu/~ metis* (1998).

[13] George Karypis and Vipin Kumar. "Multilevelk-way partitioning scheme for irregular graphs". In: *Journal of Parallel and Distributed computing* 48.1 (1998), pp. 96–129.

[14] Charles J Alpert. "The ISPD98 circuit benchmark suite". In: *Proceedings of the 1998 international symposium on Physical design*. 1998, pp. 80–85.

[15] Ricardo Maronna. "Charu C. Aggarwal and Chandan K. Reddy (eds.): Data clustering: algorithms and applications". In: *Statistical Papers* 57.2 (2016), p. 565.

[16] Tomas Barton, Tomas Bruna, and Pavel Kordik. "Chameleon 2: an improved graph-based clustering algorithm". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 13.1 (2019), pp. 1–27.

[17] Charles M Fiduccia and Robert M Mattheyses. "A linear-time heuristic for improving network partitions". In: *Papers on Twenty-five years of electronic design automation*. 1988, pp. 241–247.

[18] Tomáš Bartoň and Pavel Kordík. "Evaluation of relative indexes for multi-objective clustering". In: *Hybrid Artificial Intelligent Systems: 10th International Conference, HAIS 2015, Bilbao, Spain, June 22-24, 2015, Proceedings 10*. Springer. 2015, pp. 465–476.