Chameleon2++: An Efficient and Scalable Variant Of Chameleon Clustering

MS(Research) Thesis

By

Priyanshu Singh

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

December 2024

Chameleon2++: An Efficient and Scalable Variant Of Chameleon Clustering

A THESIS

submitted to the

INDIAN INSTITUTE OF TECHNOLOGY INDORE

in fulfillment of the requirements for

the award of the degree

of

MS(Research)

By

Priyanshu Singh

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

December 2024

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Chameleon2++: An Efficient and Scalable Variant Of Chameleon Clustering** in the fulfillment of the requirements for the award of the degree of **MS (Research)** and submitted in the **Department of Computer Science and Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from August 2021 to December 2024.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

[illegible] 7/7/2025

Signature of Student with Date
(Priyanshu Singh)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

[illegible] 7/7/2025

Signature of Thesis Supervisor with Date
(Prof. Kapil Ahuja)

Priyanshu Singh has successfully given his MS(Research) Oral Examination held on 2/7/2025

Signature of Chairperson, OEB Date:	Signature of External Examiner Date:	Signature of Thesis Supervisor Date:
Signature of PSPC Member #1 Date:	Signature of PSPC Member #2 Date:	Signature of Convener, DPGC Date:
Signature of Head of Discipline Date:		

4. The candidate has [✓ PASSED] / FAILED in the MS (RESEARCH) ORAL EXAMINATION

ORAL EXAMINATION BOARD

5. Letter grade awarded in XX 794: M.S. Thesis (Stage-4):____________________.

Name and Affiliation of the Examiners	Signature with date
1. Prof Soumyendu Raha (External) IISc Bangalore	[illegible] 02 July 2025
2. Prof. Swadesh Sahoo (Chairman) IIT Indore	[illegible] 2/7/2025
3. Prof. Kapil Ahuja (Supervisor) IIT Indore	[illegible] 2/7/2025
4. Dr. Soumi Chattopadhyay IIT Indore (DPGC Convener)	Soumi 2/7/2025
5. Dr. Chandresh Kumar Maurya IIT Indore (HOD CSE Rep.)	[illegible] 2/7/25
6.	
7.	
8.	

[illegible] 2/7/2025

(Signature of Chairman, MS (Research) Oral Examination Board with date)

Soumi 2/7/2025

(Signature of Convener, DPGC with date)

For Office Use Only

Course Units: **CPI:**

The candidate has met all the requireements for the award of the MS (Research) degree and be issued provisional degree after submission of hard bound copies of the Thesis and the No Dues Certficate.

Signature of JR/ DR/ AO, Academics Affairs (with date)

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my heartfelt gratitude to a number of persons who in one or the other way contributed by making this time as learnable, enjoyable, and bearable. First, I would like to thank my supervisor **Prof. Kapil Ahuja**, a constant source of inspiration during my work. Without his constant guidance and research directions, this research work could not be completed. His continuous support and encouragement have motivated me to remain streamlined in my research work.

I am thankful to **Prof. Aruna Tiwari** and **Prof. Pankaj Sagdeo**, my PSPC members for taking out some valuable time to evaluate my progress throughout the course. Their good comments and suggestions helped me to improve my work at various stages. I am also grateful to **DPGC** and **HOD** of Computer Science and Engineering for their help and support.

My sincere acknowledgment and respect to **Prof. Suhas Joshi** , Director, Indian Institute of Technology Indore for providing me the opportunity to explore my research capabilities at Indian Institute of Technology Indore.

I would like to express my heartfelt respect to my parents for the love, care, and support they have provided to me throughout my life.

Finally, I am thankful to all who directly or indirectly contributed, helped, and supported me.

Priyanshu Singh

Abstract

Hierarchical clustering remains a fundamental challenge in data mining, particularly when dealing with large-scale datasets where traditional approaches fail to scale effectively. Recent Chameleon-based algorithms—Chameleon2 (2019), M-Chameleon (2021), and INNGS-Chameleon (2021)—have advanced strategies but suffer from $O(n^2)$ computational complexity. Particularly in their graph generation stage due to exact k-NN computation. While tolerable on synthetic or small datasets, this quickly becomes a bottleneck for real-world datasets which are large-scale and high-dimensional.

We introduce **Chameleon2++**, a scalable extension of Chameleon2 tailored for real-world applications. Our algorithm has three parts: (1) *Graph Generation* — we integrate the Annoy algorithm instead of exact k-NN for fast approximate neighbor computation, significantly reducing graph construction time; (2) *Graph Partitioning* — we adapt the multilevel hMETIS algorithm instead of naive recursive FM bisection, which requires high-level tuning and has limited scope for synthetic datasets. hMETIS is robust to approximation-induced k-NN graph and yields higher-quality partitions with minimal configuration requirements; (3) *Merging* — we first apply a flood-fill heuristic to ensure connected, balanced components in the partitions, followed by the efficient merging criteria, both retained from Chameleon2.

These enhancements reduce the overall time complexity to $O(n \log n)$, achieving scalability without compromising clustering performance. On real-world benchmark datasets used in prior Chameleon works, Chameleon2++ delivers an average of **5%** improvement in clustering quality. This demonstrates that algorithmic efficiency and clustering quality can co-exist in large-scale hierarchical clustering.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

Ch2 Chameleon2

M-Ch M-Chameleon

INNGS-Ch INNGS-Chameleon

Ch2++ Chameleon2++

k**-NN** k-Nearest Neighbors

FM Fiduccia-Mattheyses

hMETIS Hypergraph-based Multi-level Recursive Bisection

NMI Normalized Mutual Information

ACC Accuracy

ANNS Approximate Nearest Neighbors Search

Annoy Approximate Nearest Neighbors Oh Yeah

R_{CL}, RC Relative Closeness

R_{IC}, RIC Relative Interconnectivity

Chapter 1

Introduction

Hierarchical clustering represents a cornerstone technique in data mining and machine learning, providing intuitive tree-structured representations of data relationships that are essential for exploratory data analysis, pattern recognition, and knowledge discovery. As modern applications generate increasingly large-scale and high-dimensional datasets, the computational demands of traditional hierarchical clustering algorithms have become a critical bottleneck, limiting their applicability to real-world scenarios where scalability is paramount.

The Chameleon family of algorithms has emerged as a leading approach to hierarchical clustering, distinguished by their sophisticated merging criteria that consider both inter-cluster connectivity and closeness. Recent advances in this domain include Chameleon2 (Ch2) (2019) (1), M-Chameleon (M-Ch) (2021) (2), and INNGS-Chameleon (INNGS-Ch) (2021) (3), each contributing novel merging strategies and refinements to the original framework. However, these algorithms universally suffer from $O(n^2)$ computational complexity, primarily attributed to their reliance on exact k-nearest neighbor (k-NN) computation during the graph generation phase. While this complexity remains manageable for synthetic datasets, it becomes prohibitive when applied to real-world datasets characterized by extensive feature spaces.

We introduce Chameleon2++ (Ch2++), a scalable extension of the Ch2 algorithm specifically designed to address these computational limitations while maintaining clustering quality. Our approach systematically enhances two fundamental stages of the Chameleon framework. First, instead of exact k-NN computation, we adapt the Annoy (Approximate Nearest Neighbors Oh Yeah) algorithm for approximate nearest neighbor (ANN) search, dramatically reducing graph construction time without significant quality degradation. Second, existing Ch2 uses recursive Fiduccia-Mattheyses (FM) bisection for graph partitioning, which works well on synthetic datasets due to its high configurability but requires extensive manual tuning that limits its applicability to real-world scenarios. Instead, we integrate the multilevel hypergraph Multilevel Graph Partitioning (hMETIS) algorithm for graph partitioning, which demonstrates superior robustness to approximation-induced noise in the graph structure and produces higher-quality partitions with minimal configuration requirements for both small and large-scale, high-dimensional datasets. Post graph partitioning, we retain the idea of flood-fill and merging criteria from Ch2. With these changes, Ch2++ achieves $O(n \log n)$ computational complexity compared to the original $O(n^2)$, while maintaining clustering quality.

Next we perform experiments on all the real-world benchmark datasets utilized in Ch2, M-Ch and INNGS-Ch and demonstrate that this gain in complexity is not coming at the cost of performance, instead we are slightly better than these algorithms in performance. We achieve an average gain of 5% in clustering performance across all large-scale and high-dimensional benchmark datasets. For Ch2, M-Ch and INNGS-Ch specifically we achieve consistent clustering performance improvements of 1%, 8%, and 5% over these respective algorithms. These results establish that algorithmic efficiency and clustering quality can indeed coexist in large-scale hierarchical clustering applications.

The remainder of this thesis is organized as follows. In Chapter 2, we revisit recent variants of Chameleon clustering algorithms and analyze their computational complexities and shortcomings. Chapter 3 introduces Chameleon2++, our efficient and scalable improvement to Chameleon2. In Chapter 4, we demonstrate the clustering performance comparison between Chameleon2++ and existing variants, namely

Chameleon2, M-Chameleon, and INNGS-Chameleon. Finally, in Chapter 5, 6, we conclude with a summary of our findings and directions for future research respectively.

Chapter 2

Literature Review

This chapter reviews recent variants of Chameleon clustering algorithms, namely Chameleon2, M-Chameleon, and INNGS-Chameleon. For each algorithm, we discuss its key components and their default configurations, analyze computational complexity, and identify limitations that motivate our proposed improvements. These algorithms are detailed in Sections 2.0.1, 2.0.2, and 2.0.3 respectively.

2.0.1 The Chameleon2 Algorithm

In this section, we present Ch2, a hierarchical agglomerative clustering algorithm. Ch2 operates in three key phases: graph generation, graph partitioning, and merging. The first two phases aim to partition the data into small subclusters, while the third refines these subclusters and iteratively combines them to form the final clusters using a merging criteria. These aspects are discussed in detail in the following subsections.

2.0.1.1 Graph Generation

Transforming high-dimensional datasets into sparse graphs can effectively reveal underlying patterns in the data. Here, this is achieved by using the k-Nearest Neighbor (k-NN) algorithm, which connects each node to its k most similar neighbors forming k edges per node. The edges are weighted based on the inverse of the distance between nodes, meaning shorter distances correspond to higher edge weights given by that $\frac{1}{D(N1,N2)}$.

Here, D is a distance function, usually Euclidean, and $N1$ and $N2$ are two nodes. In Ch2, a symmetrical k-NN graph is constructed, as it exhibits substantially fewer edges. Symmetrical k-NN constructs a graph in which an edge exists between two nodes only if *both* nodes belong to the k-nearest neighbors of each other. By eliminating redundant intra-cluster connections (focusing on more relevant relationships within the data), it improves clustering quality.

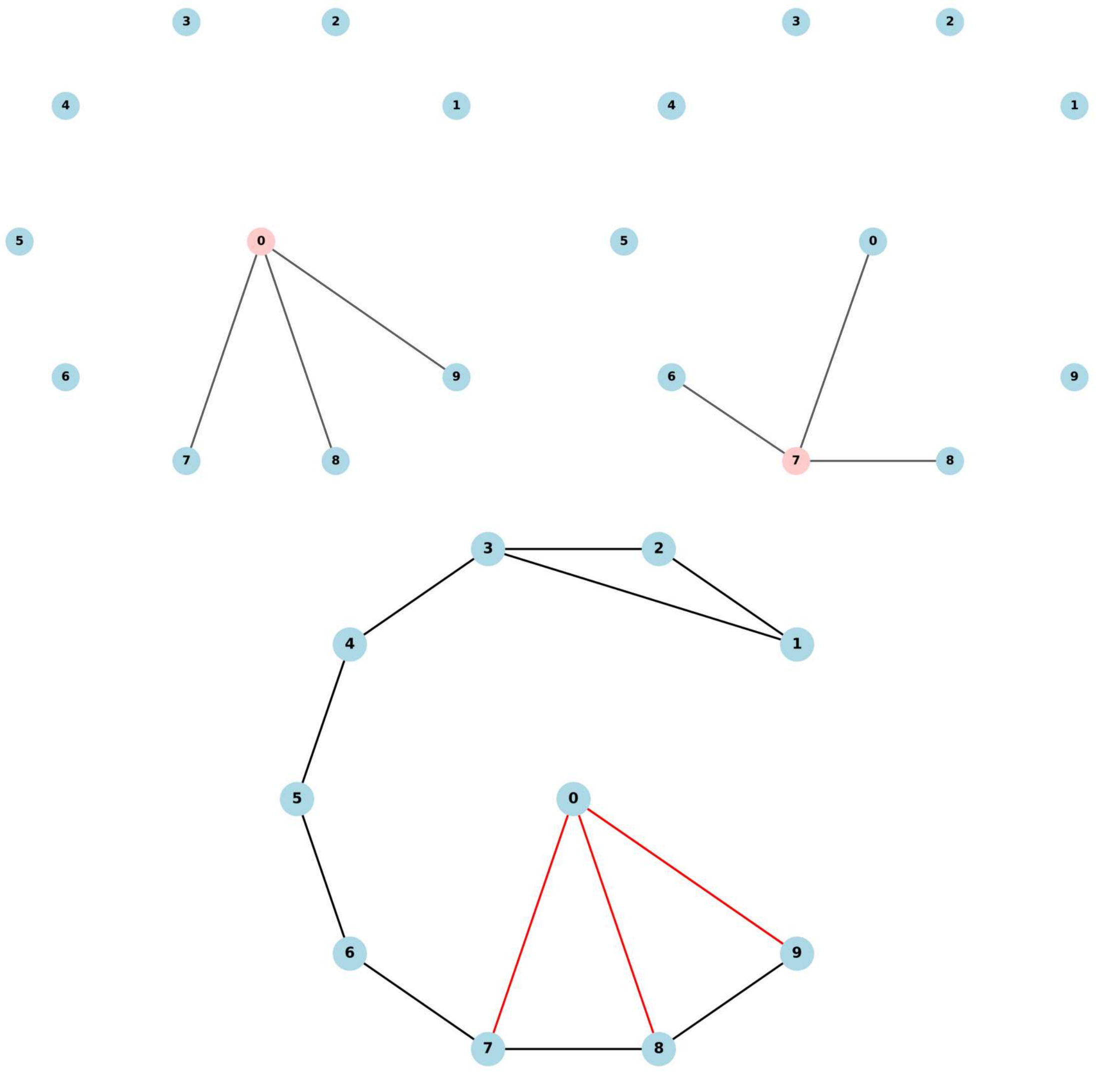

Figure 2.1: Visualization of 3-nearest neighbors: (Top left) 3-NNs for node 0, (Top right) 3-NNs for node 7, and (Bottom) Symmetrical 3-NNs graph.

Fig. 2.1 illustrates the construction of symmetrical k-NN graph for an abstract dataset, highlighting the intermediate step of identifying the k-NNs of two nodes and establishing edges only between mutual k-NNs in the final graph. Here, 0-7, 0-8, 0-9 have edges between them because they are in 3-NNs of each other. While 0-1, 0-2, 0-3, 0-4, 0-5, 0-6 do not have edges between them because although 0 is in 3-NNs of 1, 2, 4, 5, 6, vice-versa is not true (0 already has 7, 8, 9 as symmetrical 3-NNs).

The graph generation algorithm uses exact k-nearest neighbors and has a runtime complexity of $O(n^2)$, and the default value of k is $2 \cdot \ln(n)$, here n denotes the number of nodes in the graph (or dataset). The algorithm has to visit all neighbors of every item to find the nearests.

2.0.1.2 Graph Partitioning

This section summarizes the Fiduccia-Mattheyses (FM) graph partitioning algorithm as utilized in Ch2, which primarily takes a k-NN graph as input. The goal of graph partitioning is to divide a graph into multiple subclusters of roughly equal size while minimizing the edge-cut, i.e., the total number of edges (or weights) connecting nodes between different subclusters. FM here employs recursive bisection, iteratively splitting the graph into halves until each partition reaches a specified size threshold.

2.0.1.2.1 Fiduccia-Mattheyses Algorithm FM is a variant of the Kernighan-Lin (KL) algorithm (4), which we discuss below. Initially, the algorithm generates a random partition of the input graph, dividing it into two approximately equal-sized parts. Then, it computes the gain for each node, referred to as potential change in minimum cut-size if the node was moved to the opposite partition. These gains are stored in a data structure called a gain-bucket.

Fig. 2.2 illustrates an example with 4 nodes connected by certain edges, with a random red-colored cut showing the initial partition. The left side of the figure is the initial iteration, while the right side depicts the final iteration, marking the completion of the first pass. While looking at the left figure, on the left side of the partition, moving node v_2 would reduce the cut size by 2, as shown in the left figure (hence, its gain is +2), while moving node v_0 to other side would result in a gain of +1. We can similarly compute the gains on the right side of the partition.

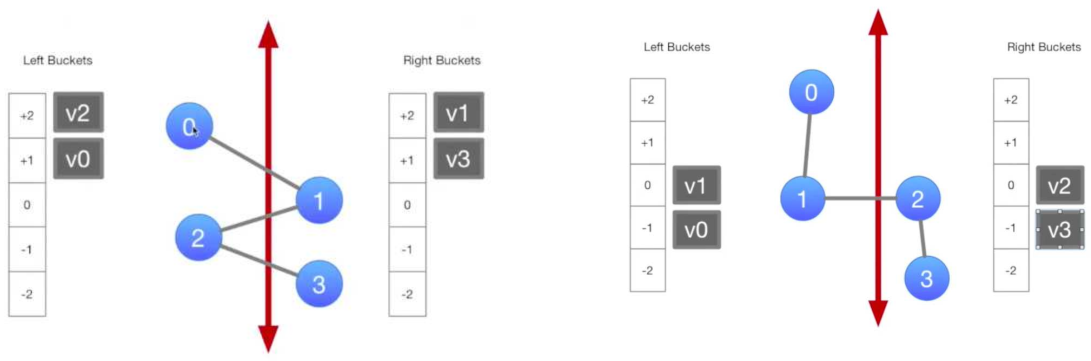

Figure 2.2: FM utilizing gain bucket data structure to select the optimal cut and update buckets for the next pass. (Left) Initial bucket values for each node. (Right) Bucket updates after a single pass.

Next, the algorithm iteratively selects the node with the highest gain from each partition and moves it to the other partition. After each move, the selected node is locked and excluded from future iterations, while the gains of its neighboring nodes are updated in the gain bucket. The algorithm continues selecting and moving nodes from the remaining unlocked set. Upon completing all node moves, the configuration with the smallest cut-size observed during the process is chosen as the final partition. This completes one pass of the algorithm. The result of one complete pass is depicted in right side of Fig. 2.2.

Once the first pass is completed, then the configuration of the smallest cut-size becomes the starting point of performing another pass on it. These multiple passes are performed because the nature of the algorithm is greedy and it may result in the algorithm getting stuck in a local minima. This continues until no further improvement is achieved or a maximum number of passes is reached. The Ch2 algorithm does not use the standard FM algorithm; instead, it employs a recursive version known as recursive FM bisection. The algorithm utilized in Ch2 requires one parameter, p_{max}, defining the maximum number of nodes in a single partition (stopping criteria). The FM bisection is repeated on each resulting partition until all clusters contain no more than p_{max} nodes.

The time complexity of the algorithm is $O(n \log m)$, where n is the number of nodes in the graph and m denotes number of partitions bounded by p_{max} nodes. The amount of time needed to bisect all graphs at a single level of recursion is always $O(n)$. Depth of the recursion depends on the desired number of partitions. To obtain m partitions of size n/m, we need to recursively bisect the graph $\log(m)$ times.

2.0.1.3 Merging

A major issue with partitioning algorithms is their tendency to create disconnected partitions while minimizing edge cuts. These widely separated partitions lack internal connectivity, which cannot be corrected in later stages, resulting in incorrect merges during the merging phase and ultimately degrading overall clustering quality. Fig 2.3 illustrates this issue. Here, to achieve a minimal edge cut, the graph is partitioned into disconnected partitions (excluded by red-borders).

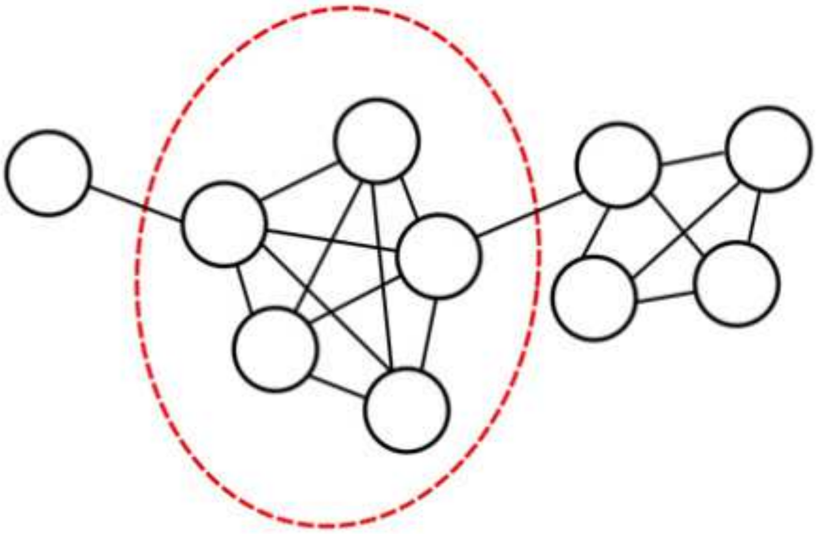

Figure 2.3: The red border separates the connected partition (within the border) and disconnected partition (outside the border).

To address this issue, Ch2 employed a local breadth-first search (BFS) based heuristic called Flood-Fill on the partitioned k-NN graph, which detects the disconnected components within the partitions. If a partition consists of multiple disconnected components, the algorithm separates them into disjoint connected partitions. This approach ensures that partitions are cohesive and do not contain disjointed substructures. By this approach, Ch2 achieves significantly better clustering quality effectively. Here, the run-time complexity of the heuristic is $O(n)$.

Next, merging outlines the final phase of the algorithm. After partitioning and flood-fill, several small sub-clusters are generated, which must be merged to form the final clusters. To achieve this most existing algorithms rely just on external properties. For example, the distance between the centers of the clusters, etc. (5). However, besides external properties, the internal properties of clusters plays a crucial role to achieve better clustering, for example density and structural similarity of the clusters.

Here, merging in Ch2 draws primary inspiration from (6) and (7) and is performed using two similarity metrics: Relative-Interconnectivity (R_{IC}) and Relative-Closeness (R_{CL}). R_{IC} measures the external interconnectivity between two clusters relative to their internal interconnectivity, while R_{CL} measures the external closeness between two clusters relative to their internal closeness. The key insight is that clusters should be merged not just based on how connected or close they are to each other, but relative to their own internal structure and density. Next, we define the mathematical formulation of these. If C_i and C_j denote ith and jth clusters, then the R_{IC} between them is defined as follows (1):

$$R_{IC}(C_i, C_j) = \begin{cases} 1, & \text{for } |E_{C_i}| \vee |E_{C_j}| = 0, \\ \frac{|E_{C_{i,j}}|}{\min\{|E_{C_i}|, |E_{C_j}|\}} \cdot \rho(C_i, C_j)^{\beta} & \text{for } |E_{C_i}| \wedge |E_{C_j}| > 0. \end{cases} \tag{2.1}$$

Above, first row signifies singleton clusters and second row takes care of all the other cases. Here, $|E_{C_{i,j}}|$ represents the number of edges between cluster C_i and C_j; $|E_{C_i}|$ denotes the number of edges within cluster C_i; and $|E_{C_j}|$ denotes the number of edges within cluster C_j. Here, ρ discourages the algorithm from merging clusters with different densities and the β parameter (with default value of 1.0) serves to modify the weight of the ρ factor. The expression for ρ is given as below (7).

$$\rho(C_i, C_j) = \frac{\min\{\overline{s}(C_i), \overline{s}(C_j)\}}{\max\{(\overline{s}(C_i), \overline{s}(C_j)\}}, \tag{2.2}$$

where,

$$\overline{s}(C_i) = \frac{1}{|E_{C_i}|} \sum_{e \in C_i} w(e). \tag{2.3}$$

Here, $w(e)$ denotes the weight of an edge e within a cluster C_i. Similarly, the formula for Relative-Closeness (R_{CL}) is given as follows:

$$R_{CL}(C_i, C_j) = \begin{cases} m_{fact} \cdot \frac{\overline{s}(C_i, C_j)}{s(C_i) + s(C_j)} & \text{for } |E_{C_i}| \vee |E_{C_j}| = 0, \\ (|E_{C_i}| + |E_{C_j}|) \cdot \frac{\overline{s}(C_i, C_j)}{s(C_i) + s(C_j)} & \text{for } |E_{C_i}| \wedge |E_{C_j}| > 0. \end{cases} \tag{2.4}$$

Here, as above, the first line captures the singleton case and the second captures all the other cases. Here, m_{fact} is a constant factor which ensures that singleton clusters obtain higher similarly value, which causes the them to merge with their neighbors in the early stages of the merging process.

Finally, at each iteration, Ch2 selects the cluster pairs which maximize S_{Ch2} given below in agglomerative order. Ultimately, the algorithm reports the clustering with the highest clustering metric at any merge stage.

$$S_{Ch2}(C_i, C_j) = R_{CL}(C_i, C_j)^{\alpha} \cdot R_{IC}(C_i, C_j), \tag{2.5}$$

where, α is a user-defined parameters with default value of 2.0 respectively.

The naive merging process would have complexity of $O(m^3)$ by examining all cluster pairs at each step. By using a priority queue to store similarities between cluster pairs, we can reduce this complexity to $O(m^2 \log m)$ with m merging steps, where finding the most similar pair takes $O(1)$ time and queue operations take $O(\log m)$ time.

Phase	**Algorithm**	**Complexity**
Graph Generation	Exact k-NN (Symm.)	$O(n^2)$
Graph Partitioning	FM Bisection (Recursive)	$O(n \log m)$
Merging	Flood-Fill — $R_{CL}^{\alpha} \cdot R_{IC}$	$O(n) + O(m^2 \log m)$

Table 2.1: Ch2 Algorithm Phases and Complexities.

The final complexity of Ch2 is summarized in Table 2.1, i.e., $O(n^2 + n \log m + n + m^2 \log m)$. As discussed before m would be significantly smaller than n, (i.e., $m \ll n$) therefore, the overall complexity is bounded by $O(n^2)$.

2.0.1.3.1 Shortcomings of Ch2 Ch2 faces several critical limitations that impact its scalability and practical applicability. The most significant bottleneck is the exact k-nearest neighbor computation during graph generation, which becomes computationally prohibitive for large-scale and high-dimensional datasets with $O(n^2)$ complexity. In the subsequent section, our algorithm (Ch2++) uses approximate k-NN, reducing the complexity to $O(n \log n)$.

The partitioning phase further exacerbates scalability issues through its reliance on naive recursive FM bisection, whose performance degrades as data size increases. Additionally, the algorithm appears more suited to small, low-dimensional synthetic datasets that allow extensive manual configuration. While the authors employed numerous configurations in their experiments, the paper lacks systematic disclosure of parameter variations, making result replication difficult due to insufficient experimental methodology and limiting its applicability for automated clustering. To address this, we introduce hMETIS for graph partitioning, which better absorbs approximation errors from the previous stage. We achieve all this with our algorithm without any loss in clustering performance, while providing a systematic approach for parameter configurations that makes results easier to replicate, as discussed in later sections.

2.0.2 The M-Chameleon Algorithm

In this section, we present M-Chameleon (M-Ch), a hierarchical agglomerative clustering algorithm. M-Ch operates in three key phases: graph generation via k-nearest neighbors; graph partitioning, here they identify the sub-clusters using mutual k-nearest neighbors for initial clustering; and merging, which are discussed below, respectively.

2.0.2.1 Graph Generation

M-Ch begins by constructing an asymmetric k-nearest neighbors graph where an edge exists between two nodes if one node belongs to the k-nearest neighbors of the other. Unlike traditional symmetric k-NN graphs used in Chameleon variants, this asymmetric construction allows for more flexible connectivity patterns. The default value for k is set to 2. Here, the run-time complexity is similar to that of Ch2's graph generation, i.e., $O(n^2)$, where n is the number of nodes.

2.0.2.2 Graph Partitioning

Following the initial k-NN graph generation, M-Ch builds a mutual k-nearest neighbors graph G_{MKNN} on top of the initial k-NN graph, ensuring symmetric relationships as discussed in Ch2. The connected components generated from G_{MKNN} serve as initial sub-clusters, effectively replacing the traditional graph partitioning phase. To improve algorithmic efficiency, singleton clusters (containing only one node) are immediately merged with the cluster containing their nearest neighbor. The run-time complexity remains $O(n^2)$, where n is the number of nodes.

2.0.2.3 Merging

Here, in merging it is quite related on a higher level to Ch as discussed before instead being based on the average weights of intra-cluster and inter-cluster edges, M-Ch relies on relative closeness (RC) and relative interconnectivity (RI) metrics computed from graph bisection, which is computationally more expensive than the approach discussed in Ch2. If C_i and C_j denote ith and jth clusters, then the RI between them is defined as:

$$RI(C_i, C_j) = \frac{|EC(C_i, C_j)|}{\frac{|EC(C_i)|+|EC(C_j)|}{2}} \tag{2.6}$$

where $|EC(C_i, C_j)|$ is the sum of the weight of the edges that straddle clusters C_i and C_j. $|EC(C_i)|$ is the sum of weights of the edges crossing a min-cut bisection that splits the cluster C_i into two roughly equal parts. In the same way, $|EC(C_j)|$ is the sum of weights of the edges crossing a min-cut bisection that splits the cluster C_j into two roughly equal parts. Similarly, the formula for RC is defined as:

$$RC(C_i, C_j) = \frac{SEC(C_i, C_j)}{\frac{|C_i|}{|C_i|+|C_j|}SEC(C_i) + \frac{|C_j|}{|C_i|+|C_j|}SEC(C_j)} \tag{2.7}$$

where $SEC(C_i, C_j)$ is the average weight of edges that straddle clusters C_i and C_j. $SEC(C_i)$ and $SEC(C_j)$ are the average weight of the edges cutting of the min-cut bisector of clusters C_i and C_j respectively. $|C_i|$ and $|C_j|$ are the number of vertices in each cluster.

Finally, the similarity score of M-Ch algorithm merging sub-clusters is measured according to the above relative connectivity and relative closeness, as shown in:

$$Sim(C_i, C_j) = RI(C_i, C_j) \times RC(C_i, C_j)^{\alpha} \tag{2.8}$$

where α is a balance parameter that adjusts these two indicators. If $\alpha > 1$, it means that relative closeness has more influence on similarity; on the contrary, if $\alpha < 1$, it gives higher importance to the relative interconnectivity. In M-Ch, $\alpha = 1$ is a user-defined parameter. M-Ch merges the largest similarity cluster pair to form a true cluster.

Additionally, M-Ch incorporates the MC modularity criterion from social network analysis to automatically determine the optimal clustering result, eliminating the need for manual specification of the final cluster count. The algorithm iteratively merges the two clusters with the highest combined RI and RC values while monitoring the MC modularity to identify the best clustering configuration. The MC modularity is defined as:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{2.9}$$

where A_{ij} is the adjacency matrix element between nodes i and j, k_i and k_j are the degrees of nodes i and j respectively, m is the total number of edges in the graph, c_i and c_j are the community assignments of nodes i and j, and $\delta(c_i, c_j)$ is the Kronecker delta function that equals 1 if $c_i = c_j$ and 0 otherwise.

The change in modularity when merging two clusters C_i and C_j is given by:

$$\Delta Q = \frac{1}{2m} \left[2e_{ij} - \frac{2k_i k_j}{2m} \right] \tag{2.10}$$

where e_{ij} is the number of edges between clusters C_i and C_j, and k_i, k_j are the sum of degrees of all nodes in clusters C_i and C_j respectively.

The merging process continues in agglomerative manner, reporting the optimal clustering configuration where modularity Q holds its maximum value. At each step, the algorithm evaluates:

$$\text{Merge Score} = Sim(C_i, C_j) \times \Delta Q_{ij} \tag{2.11}$$

The pair of clusters with the highest merge score is selected for merging, ensuring both structural similarity and modularity optimization.

Phase	**Algorithm**	**Complexity**
Graph Generation	Exact k-NN (Asymm.)	$O(n^2)$
Graph Partitioning	Mutual k-NN (sub-clusters)	$O(n^2)$
Merging	$RC^{\alpha} \cdot RI$	$O(m^3)$

Table 2.2: M-Ch Algorithm Phases and Complexities.

The overall complexity of M-Ch is summarized in Table 2.2, yielding $O(n^2 + n^2 + m^3)$. Since the number of initial sub-clusters m is significantly smaller than n (i.e., $m \ll n$), the overall complexity is dominated by and bounded by $O(n^2)$.

2.0.2.3.1 Shortcomings of M-Ch M-Ch exhibits prohibitively high computational complexity when processing large-scale high-dimensional data, particularly in both graph generation and partitioning phases, which results in poor scalability. Additionally, the algorithm lacks explicit noise handling mechanisms, leading to degraded clustering accuracy on datasets containing significant noise or outliers. We address the scalability issues by reducing the complexity of the graph generation phase to $O(n \log n)$ while improving clustering performance through approximate k-NN in our proposed algorithm. Furthermore, for sub-cluster identification, we introduce hMETIS which better absorbs approximation errors from the previous stage and yields high quality partitions, indirectly improving robustness to noise.

2.0.3 The INNGS-Chameleon Algorithm

In this section, we present INNGS-Chameleon (INNGS-Ch), a hierarchical agglomerative clustering algorithm that combines improved natural neighbor graph construction with probabilistic sub-cluster identification. INNGS-Ch operates in three key phases: improved natural neighbor graph construction, probabilistic sub-cluster generation, and hierarchical merging. These aspects are discussed in detail in the following subsections.

2.0.3.1 Graph Generation

INNGS-Ch constructs an asymmetrical k-NN graph similar to M-Ch called as a Natural Neighbor Graph, where the value of k is adaptively determined through a stability-based approach. Here, the authors introduce a ”stable state” concept that monitors the connected components of the graph during construction. The algorithm iteratively searches for the k-nearest neighbors of each data point and evaluates the number of connected components m in the resulting graph. Once the stable state is achieved, if $c \leq m$ (where c is the desired number of clusters), the current configuration is accepted; otherwise, k is incremented by 1 (initially started from 2) and the process continues until $c \leq m$.

Notably, this approach requires specifying the number of clusters c, the authors argue that this does not introduce additional parametric complexity since the underlying Chameleon algorithm already requires this parameter as input. The runtime complexity is $O(kn^2)$ where k is the adaptive neighborhood size (a constant), which in practice is often bounded by the desired cluster count c, hence $O(n^2)$, where n is the dataset size.

2.0.3.2 Graph Partitioning

INNGS-Ch employs a novel approach for generating initial sub-clusters by adapting the K-Means++ initialization strategy. INNGS-Ch directly generates sub-clusters through probabilistic center selection and graph-based distance assignment. The sub-cluster generation process begins by selecting cluster centers using a modified K-Means++ approach to ensure uniform distribution across the dataset. The first cluster center is randomly selected, followed by iterative selection of subsequent centers based on probability distributions proportional to the squared shortest distance from existing centers. Specifically, the probability of selecting a sample x as the next cluster center is given by $\frac{D_x^2}{\sum_{x \in X} D_x^2}$, where D_x represents the shortest distance between sample x and the nearest existing cluster center. The algorithm continues this probabilistic selection until $n/10$ cluster centers are chosen, where n is the total number of samples in the dataset.

Once all cluster centers are selected, the algorithm calculates shortest path distances between all non-center objects and each cluster center using the improved natural neighbor graph G_{INN}. Each non-cluster center object is then assigned to its closest cluster center based on these graph-based distances, forming the initial sub-clusters that serve as input for the subsequent hierarchical merging phase.

The time complexity of the sub-cluster generation phase is $O(n^2)$, where the dominant cost arises from computing shortest path distances between n data points and $n/10$ cluster centers on the sparse improved natural neighbor graph.

2.0.3.3 Merging

Here, similar to M-Ch, merging relies on the metrics discussed in 2.0.2.3. Merging is performed in an agglomerative manner until c number of clusters are left, where c is the desired number of clusters. This value is taken directly from the dataset if it is labeled, or else as a user-defined parameter.

Here, α is a user-defined parameters with default values of 1.0. Similar to M-Ch, the naive merging process has time complexity of $O(m^3)$.

Phase	**Algorithm**	**Complexity**
Graph Generation	Improved Natural Neighbor	$O(n^2)$
Graph Partitioning	K-Means++ — Graph Distance	$O(n^2)$
Merging	$RC^{\alpha} \cdot RI$	$O(m^3)$

Table 2.3: INNGS-Ch Algorithm Phases and Complexities.

The final complexity of INNGS-Ch is summarized in Table 2.3, i.e., $O(n^2 + n^2 + m^3)$. As discussed before m would be significantly smaller than n, (i.e., $m \ll n$) therefore, the overall complexity is bounded by $O(n^2)$.

2.0.3.3.1 Shortcomings of INNGS-Ch INNGS-Ch suffers from high computational complexity when processing large-scale high-dimensional data. Additionally, the graph generation phase requires the number of final clusters (c) as a parameter, which necessitates manual intervention or user-defined input for unlabeled datasets, thereby limiting its applicability for automated clustering across diverse datasets without prior knowledge of the cluster structure. Our algorithm (Ch2++) addresses these limitations by using approximate k-NN, reducing the complexity to $O(n \log n)$.

Chapter 3

Chameleon2++: Algorithm Design

This chapter introduces our proposed Chameleon2++ (Ch2++) algorithm. Here, we describe the Ch2++ algorithm including its phases of graph generation, graph partitioning and merging in the respective subsections below.

3.0.1 Graph Generation: Approximate k-NN

As discussed earlier, traditional exact k-NN graphs require exhaustive search to find the k-nearest neighbors of each data point by evaluating its Euclidean distance with all remaining data points, spanning the entire search space. In contrast, approximate k-NN graphs utilize approximate nearest neighbor (ANN) search algorithms, which reduce the search space by eliminating irrelevant data points. For large-scale datasets, exact k-NN computation becomes computationally prohibitive, making approximate methods both practical and necessary. The objective is to limit computation to a select subset of data points, particularly those in close proximity to the target point (8), thereby improving efficiency at the cost of potentially missing some neighbors. In Ch2++, we incorporate the same symmetrical k-NN graph approach from Ch2 and generate a symmetrical approximate k-NN graph using Annoy (9), publicly available at [1].

3.0.1.0.1 Annoy (Approximate Nearest Neighbors Oh Yeah) Annoy is one of the simple and effective algorithm in the category of approximate nearest neighbors search (9).

[1]https://github.com/spotify/Annoy

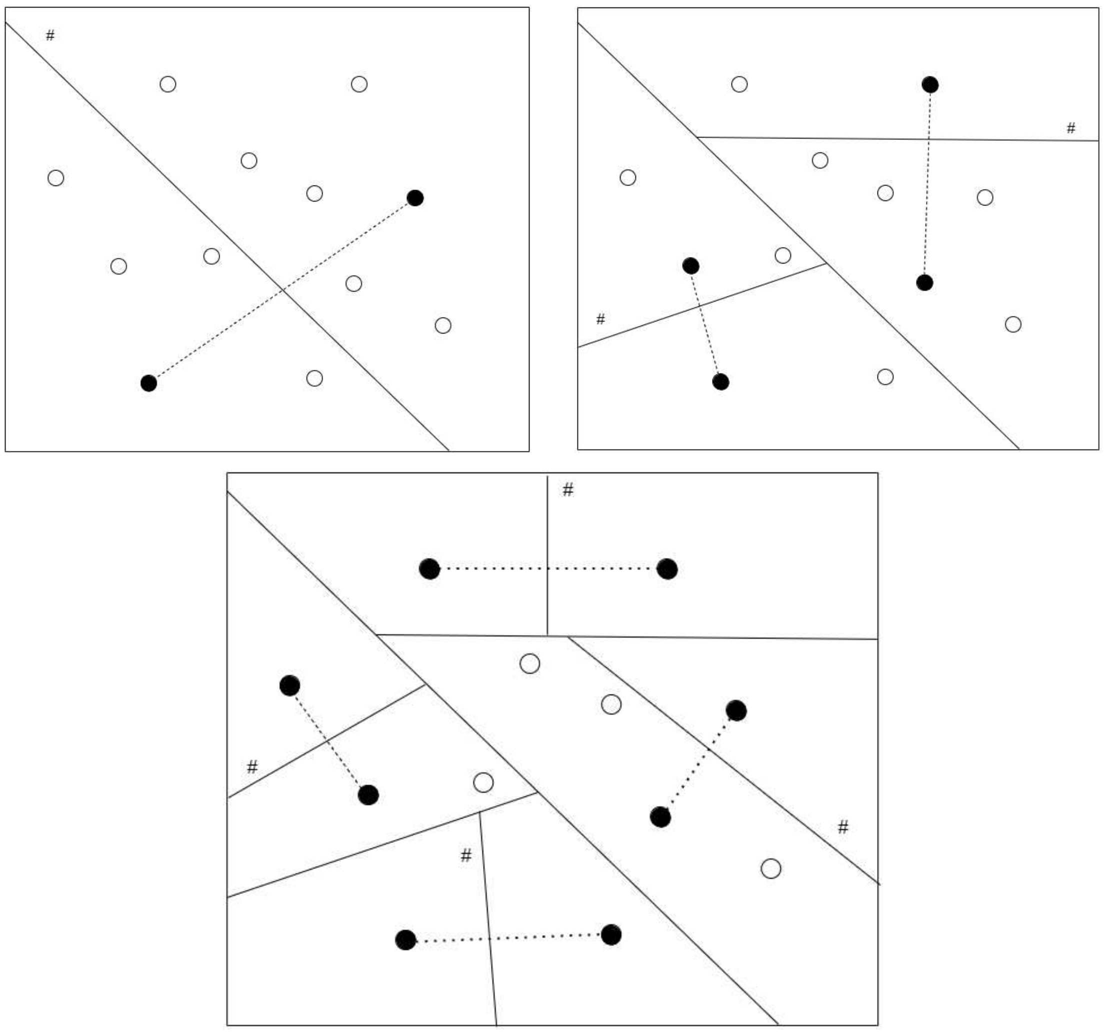

Figure 3.1: Initial recursive bi-partitioning of search-space in Annoy.

Initially, two random data points are selected, and a binary spaced partitioning is performed based on a selected hyperplane. The hyperplane here is defined as the perpendicular bisector of the line segment connecting the two selected data points. This process is recursively performed, where the data points are chosen from the subset of data being partitioned at each step. This process continues until each partition has $\leq l$ items (here, l is leaf-size parameter). This process is illustrated in Fig. 3.1.

Next, a Random Projection tree (RP-tree) is built, which serves as the indexing mechanism for nearest neighbor queries. The internal nodes here represent the hyperplane and the leaf-nodes denote the search subspace. Let q be the query data point for which the k-nearest neighbors are required. Starting with the root of a tree, the path to that child of the root is traversed which is closer to q. This process is repeated until we reach a leaf node. The leaf node gives the k-nearest neighbors for q. Building of a sample RP-Tree is shown in Fig. 3.2. Here, the number inside the node denotes the count of data points present in that subspace.

When we perform the initial recursive bi-partitioning of the search space, points are chosen at random. To improve the accuracy and search performance of the algorithm this partitioning is performed in-parallel, with different sets of initial random points, and subsequently multiple RP-trees are built i.e., a forest with (t) tress. The search is then conducted simultaneously across all trees using a priority queue-based traversal. Finally, the union of all the data points obtained from the leaf nodes of all the trees are taken into consideration (after removing duplicates). This gives us the final search subspace for the k-nearest neighbors for q.

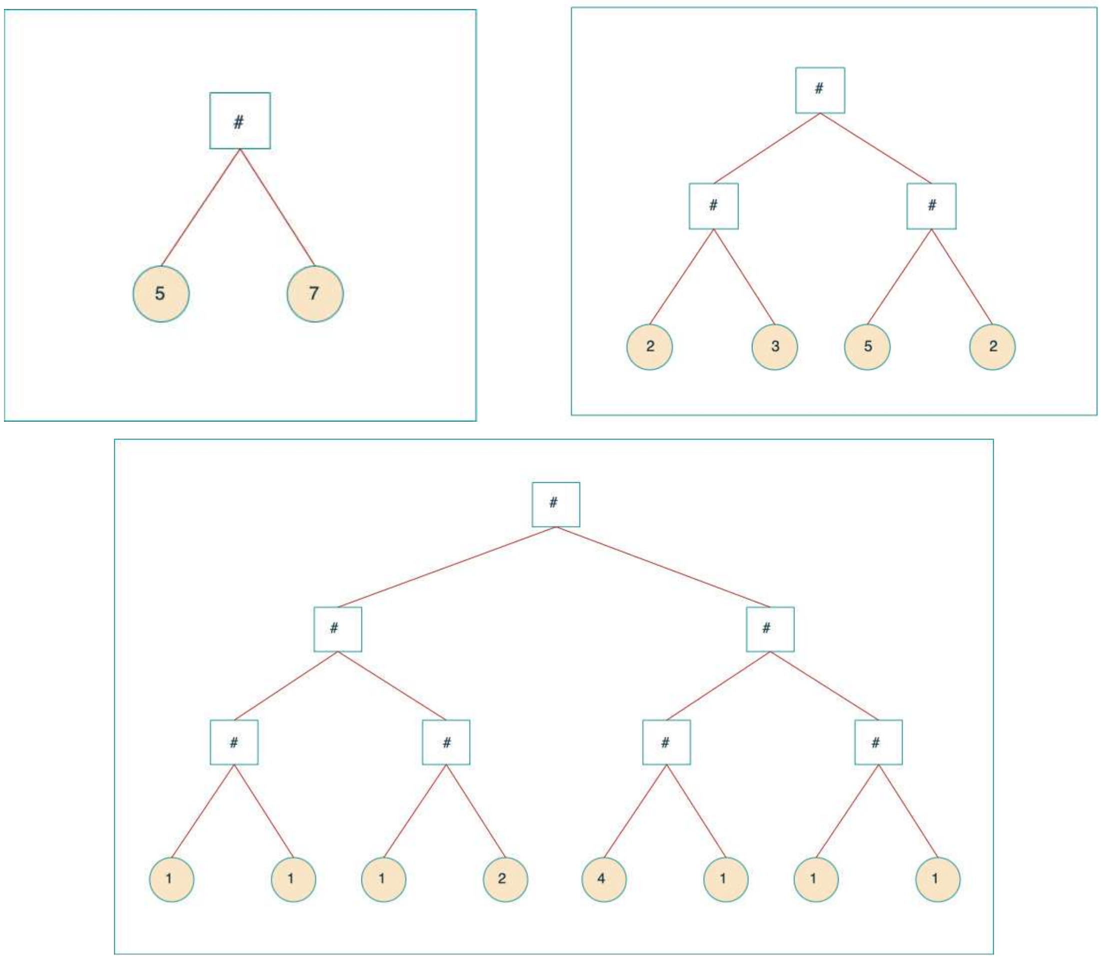

Figure 3.2: Building the tree-based index for searching in Annoy.

The time complexity for index construction in Annoy is $O(t \cdot n \log n)$, where n is the number of points in the dataset and t is the number of trees in the forest (which is a constant). For searching, the time complexity is $O(n^p \log n)$, where $p < 1$, typically around 0.5, making it sublinear in practice. Hence, the overall complexity comes out to be $O(n \log n)$.

3.0.2 Graph Partitioning: hMETIS

Since Ch2++ utilizes approximate k-NN graphs, we employ hMETIS for partitioning due to its multilevel approach, which provides better robustness compared to the naive recursive FM bisection used in Ch2. The robust nature of hMETIS allows it to better absorb errors inherent in the approximate graph structure, resulting in higher-quality partitions and improved overall clustering performance. Furthermore, given our extended scope for large-scale datasets, as emphasized in the original Ch2 study, hMETIS demonstrates superior scalability with growing dataset sizes (1).

hMETIS, developed in (10), is a multilevel partitioning algorithm designed for hypergraphs where edges can connect more than two nodes, as shown in Fig. 3.3. The hMETIS algorithm consists of three key phases: coarsening, initial partitioning, and uncoarsening (refinement) (11), as illustrated in the Fig. 3.4.

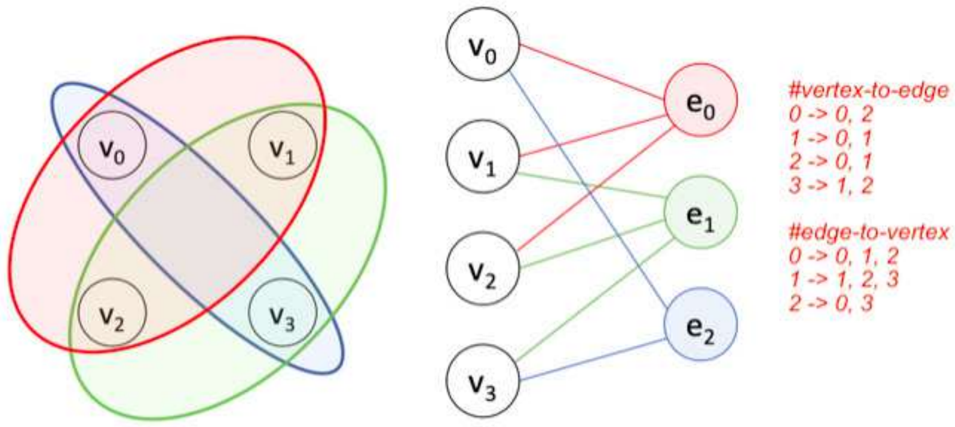

Figure 3.3: The figure on the left illustrates a hypergraph, while the figure on the right depicts its equivalent bipartite representation.

1. **Coarsening Phase** - In this phase, hMETIS iteratively creates a sequence of smaller hypergraphs by collapsing nodes and hyperedges. The algorithm uses various matching schemes to identify sets of nodes to be combined, such as edge coarsening, hyperedge coarsening, or modified hyperedge coarsening (12). For example, it combines the nodes which have large number of edges between them. This process continues until the hypergraph is sufficiently small for initial partitioning.

2. **Initial Partitioning Phase** - Once the hypergraph is coarsened to a manageable size, hMETIS applies a simple partitioning algorithm (such as KL) to create an initial partition. This step is crucial as it provides a starting point for the subsequent refinement process.

3. **Uncoarsening and Refinement Phase** - In this phase, hMETIS gradually

expands the partitioned hypergraph back to its original size. At each level of uncoarsening, the algorithm applies refinement techniques to improve the partition quality. The refinement process in hMETIS is similar to FM, but with important distinctions to handle hyperedges such as below.

- The gain calculation for moving a node considers all hyperedges it belongs to, not just binary edges.
- The gain bucket structure requires modification to accommodate hyperedge gain updates. The data structure implementation necessitates multiple arrays to efficiently manage gain calculations.
- The algorithm may also perform multiple iterations, allowing moves that temporarily increase the cut size to escape local optima.

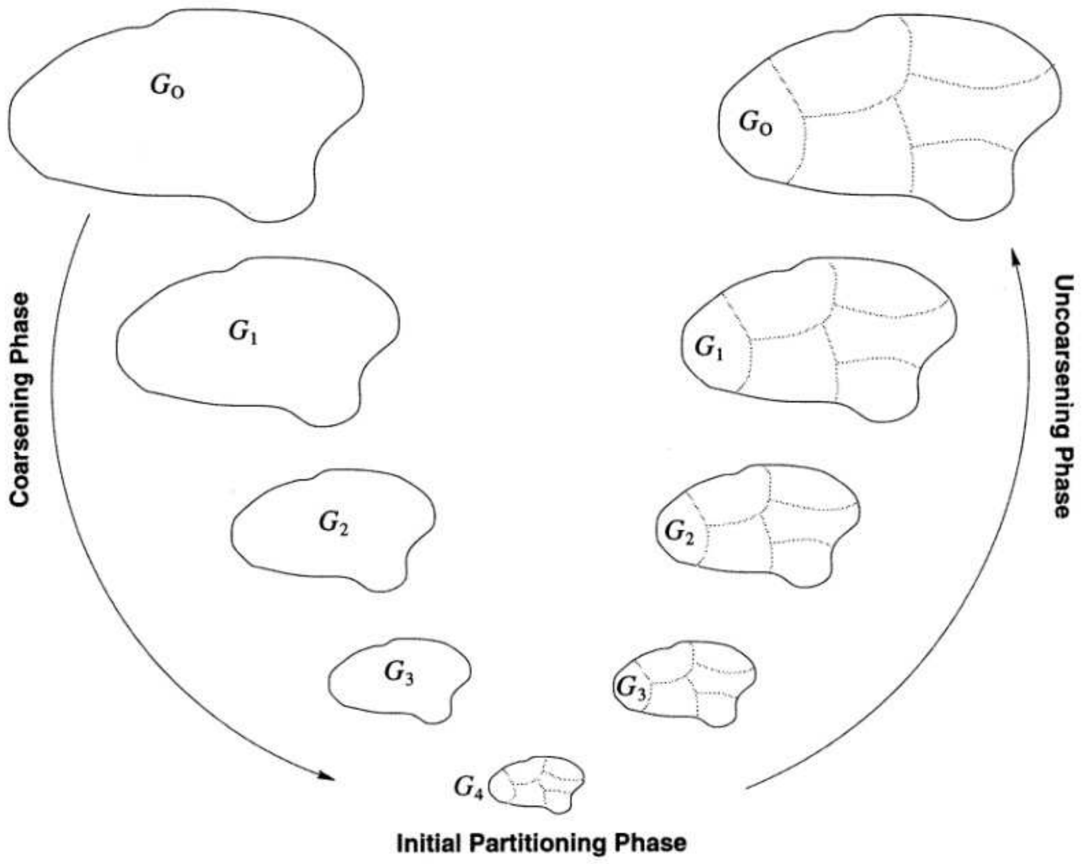

Figure 3.4: The various phases of hMETIS, a multilevel partitioning algorithm.

hMETIS additionally employs several optimization techniques such as V-cycle

Refinement, Adaptive Coarsening and Balancing Constraints etc. (13), to improve both the quality of partitions and the algorithm's runtime.

hMETIS has a runtime complexity of $O(n + m \log m)$, where n denotes the number of nodes and m is the number of partitions. We set $m = \sqrt{n}/2$ in our implementation, hence ($m \ll n$). Note that hMETIS can obtain m clusters through successive bisection with complexity $O(n \log(n/m))$ which is bounded by $O(n \log n)$, but we use the faster multilevel m-way partitioning algorithm that achieves $O(n + m \log m)$ complexity.

3.0.3 Merging

Next, for partition refinement, we employ the flood-fill algorithm from Ch2, which operates in $O(n)$ time.

Atlast, we adopt the merging criteria from Ch2 as discussed in the previous section. This phase has time complexity $O(m^2 \log m)$ where m represents the number of partitions obtained after flood-fill.

Finally, the Ch2++ base similarity combines both relative interconnectivity and relative closeness, similar to Ch2. Hence, at each iteration Ch2++ selects the cluster pairs which maximizes S_{Ch2++} for merging in agglomerative order. Ultimately, the algorithm reports the clustering with the highest clustering metric at any merge stage.

$$S_{Ch2++}(C_i, C_j) = R_{CL}(C_i, C_j)^{\alpha} \cdot R_{IC}(C_i, C_j), \tag{3.1}$$

Here, R_{CL} and R_{CL} are defined in (2.4) and (2.1). Further α and β are user-defined parameters with default values of 2.0 and 1.0, respectively.

3.0.3.0.1 Complexity Analysis The complexity analysis for each phase is presented in the following Table 3.1.

Phase	**Algorithm**	**Complexity**
Graph Generation	Approx. k-NN (Annoy)	$O(n \log n)$
Graph Partitioning	hMETIS	$O(n + m \log m)$
Merging	Flood-Fill — $R_{CL}^{\alpha} \cdot R_{IC}$	$O(n) + O(m^2 \log m)$

Table 3.1: Ch2++ Algorithm Phases and Complexities.

With these modifications, the overall complexity of Ch2++, as shown in Table 3.1, is $O(n \log n + m^2 \log m)$. Given that the number of initial sub-clusters is significantly smaller than the number of data points (i.e., $m \ll n$), the overall complexity is dominated by and bounded by $O(n \log n)$, which provides efficient scalability for large-scale clustering and surpasses the complexity of recent Chameleon variants.

Chapter 4

Results

In this section, we present a comprehensive evaluation of Ch2++ against three recent advancements in Chameleon clustering variants: Ch2, M-Ch, and INNGS-Ch. Our evaluation is conducted on large-scale, high-dimensional real-world datasets previously utilized by these recent works to demonstrate the robustness and scalability of our approach. The results are organized into four subsections: Section 4.0.1 presents the parametric configurations of Ch2++, Section 4.0.2 presents the performance comparison with Ch2, Section 4.0.3 evaluates Ch2++ against M-Ch, and Section 4.0.4 compares our method with INNGS-Ch.

4.0.1 Ch2++ Parametric Configurations

Here, we present the parameter values for Ch2++. The defaults are listed in Table4.1. Column 1 lists the parameters. Column 2 provides the parameter descriptions. Column 3 shows the default values used in our implementation.

As evident from the table, for experimentation, we used 12 combinations of k for k-NN in Annoy (derived from $r \in \{1, 2, 4, 8\}$ and $\log \in \{\ln, \log_{10}, \log_2\}$), while keeping the remaining parameters constant as listed in the table. We report the best clustering performance obtained across any of the 12 combinations for each dataset.

Table 4.1: Default Ch2++ Parameters. Here, n = dataset size).

Parameter	**Description**	**Value**
k	Approximate k-NN	$r \cdot \log n$
		$r \in \{1, 2, 4, 8\}$
		$\log \in \{\ln, \log_{10}, \log_2\}$
t	#Trees (Annoy)	k
l	Leaf-Size (Annoy)	k
m	#Partitions (hMETIS)	$\sqrt{n}/2$
m_{fact}	Small Clusters	10^3
α	R_{CL_2} Priority	2.0
β	R_{IC_2} Priority	1.0

4.0.2 Comparison with Chameleon2

For comparison with Ch2, we employ the Normalized Mutual Information (NMI) metric, which is defined as follows. NMI measures the agreement between clustering labels and ground truth, normalized to the range [0,1]:

$$\text{NMI} = \frac{2\, I(X;Y)}{H(X) + H(Y)}. \tag{4.1}$$

where $I(X;Y)$ is the mutual information between true labels X and predicted labels Y, and $H(X)$, $H(Y)$ are the entropies of X and Y respectively. NMI is symmetric, normalized, and insensitive to cluster-size imbalance, making it particularly suitable for evaluating clustering algorithms on large-scale, high-dimensional real-world datasets with varying cluster distributions.

Although they have experimented with both small toy problems and real-world datasets, the focus of this work is on the latter. Hence, we compare against large-scale and high-dimensional datasets.

Table4.2 presents the clustering performance comparison between Ch2++ and Ch2. Column 1 lists the dataset names. Column 2 provides the dataset size (n). Column 3 shows the number of features (d). Column 4 presents the NMI values for Ch2++. Column 5 shows the NMI values for Ch2. The best value for each dataset is highlighted in bold.

Table 4.2: Ch2++ vs Ch2 clustering quality comparison: NMI values of clusterings produced with the best configuration found for each dataset.

Dataset	n (size)	d (features)	Ch2++	Ch2
pendigits	10,992	16	**0.88**	0.87
cytof.h2	31,721	32	0.95	**0.98**
cytof.h1	72,463	32	0.95	**0.98**
cytof.one	81,747	13	**0.88**	**0.88**
mnist	70,000	784	0.80	**0.82**
olivetti faces	400	4,096	**0.86**	0.77
AVG.	–	–	**0.89**	0.88

As evident from the table, Ch2++ demonstrates superior performance on 2 datasets, equal performance on 1 dataset, and slightly lower performance on 3 datasets. Overall, Ch2++ achieves an average NMI of 0.89 compared to Ch2's 0.88, representing an improvement of **1%** while maintaining competitive performance across all tested datasets.

4.0.3 Comparison with M-Chameleon

M-Ch uses Accuracy as a metric for clustering performance evaluation. Accuracy (ACC) measures the fraction of correctly assigned labels after optimal label matching using the Hungarian algorithm:

$$\text{ACC} = \frac{1}{n}\sum_{i=1}^{n}\delta\big(X_i,\ \text{map}(Y_i)\big). \tag{4.2}$$

where the indicator function δ is defined as:

$$\delta(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

and X_i is the true label, Y_i is the predicted label, n is the total number of samples, and $\text{map}(\cdot)$ is the optimal permutation of cluster IDs. ACC ranges from 0 to 1, with higher values indicating better clustering performance.

The comparison of our algorithm with M-Ch on real-world datasets is presented in Table 2, which are high-dimensional in nature. These datasets have been sourced from the UCI repository and represent the same benchmark datasets utilized by M-Ch in their original evaluation.

Here in Table4.3, we present the clustering accuracy comparison between Ch2++ and M-Ch on real-world UCI datasets. Column 1 lists the dataset names. Column 2 provides the dataset size (n). Column 3 shows the number of features (d). Column 4 presents the ACC values for Ch2++. Column 5 shows the ACC values for M-Ch. The best value for each dataset is highlighted in bold.

Table 4.3: Ch2++ vs M-Ch clustering quality comparison: ACC values of clusterings produced with the best configuration found for each UCI dataset.

Dataset	n (size)	d (features)	Ch2++	M-Ch
soybean	47	35	**0.96**	0.89
iris	150	4	**0.96**	0.90
wine	178	13	0.72	**0.73**
seeds	210	7	**0.88**	0.73
sonar	208	60	**0.59**	0.57
iono	351	34	**0.77**	0.69
balance-scale	625	4	**0.69**	0.65
wil	2,000	7	**0.95**	0.89
AVG.	–	–	**0.82**	0.76

As evident from the table, Ch2++ outperforms M-Ch on most of the datasets, specifically outperforming M-Ch on 7 datasets and performing slightly worse on 1 dataset. Ch2++ achieves an average accuracy of 0.82 compared to M-Ch's 0.76, representing a substantial improvement of **8%**.

4.0.4 Comparison with INNGS-Chameleon

The clustering quality metric used for this comparison is Accuracy (ACC), which was defined in Section 4.0.3. The comparison of Ch2++ with INNGS-Ch is conducted on real-world, large-scale, and high-dimensional datasets from the

UCI repository, representing the same benchmark datasets utilized by INNGS-Ch in their original evaluation.

Table 4.4 presents the clustering accuracy comparison between Ch2++ and INNGS-Ch on real-world UCI datasets. Column 1 lists the dataset names. Column 2 provides the dataset size (n). Column 3 shows the number of features (d). Column 4 presents the ACC values for Ch2++. Column 5 shows the ACC values for INNGS-Ch. The best value for each dataset is highlighted in bold.

Table 4.4: Ch2++ vs INNGS-Ch clustering quality comparison: ACC values of clusterings produced with the best configuration found for each UCI dataset.

Dataset	n (size)	d (features)	Ch2++	INNGS-Ch
soybean	47	35	**0.96**	**0.96**
iris	150	4	**0.96**	0.95
balance-scale	625	4	**0.69**	0.57
wil	2,000	7	0.95	**0.96**
ecoli	336	8	**0.78**	0.75
dermatology	366	34	**0.58**	0.52
heart	270	13	0.54	**0.62**
pima	768	8	0.66	**0.68**
yeast	1,484	8	**0.49**	0.39
glass	214	10	**0.76**	0.75
AVG.	–	–	**0.74**	0.72

As evident from the results, Ch2++ performs better on 6 datasets, achieves equal performance on 1 dataset, and performs slightly worse on 3 datasets. Overall, Ch2++ achieves an average accuracy of 0.74 compared to INNGS-Ch's 0.72, representing an improvement of **3%**.

4.0.5 Summary

As evident from the comprehensive evaluation, our choice of approximate k-NN with robust graph partitioning via METIS does not lead to deterioration in clustering performance. In fact, Ch2++ demonstrates slightly better performance across the benchmarks. Additionally, our algorithm is computationally more efficient, achieving a complexity of $O(n \log n)$ compared to the $O(n^2)$ complexity of all three algorithms discussed above (Ch2, M-Ch, and INNGS-Ch).

Chapter 5

Conclusion

In this work, we have presented Chameleon2++, the first Chameleon-based hierarchical clustering algorithm to break the $O(n^2)$ complexity barrier while maintaining superior clustering quality. Our algorithm achieves $O(n \log n)$ computational complexity through strategic enhancements across two main algorithmic stages: replacing exact k-NN computation with Annoy's approximate nearest neighbor search for efficient graph construction, and adapting multilevel hMETIS for robust graph partitioning while retaining flood-fill and efficient Chameleon2 merging criteria.

Our comprehensive experimental evaluation demonstrates consistent performance improvements across all three recent Chameleon variants: 2% over Chameleon2, 8% over M-Chameleon, and 5% over INNGS-Chameleon on benchmark datasets. These results establish that algorithmic efficiency and clustering quality can coexist in hierarchical clustering, making Chameleon2++ viable for large-scale and high-dimensional applications.

Chameleon2++ demonstrates that approximate algorithms can be effectively integrated into sophisticated clustering frameworks without sacrificing performance, providing a template for enhancing other computationally intensive clustering algorithms.

Chapter 6

Future Work

Several promising directions emerge from this work. First, integrating advanced approximate nearest neighbor search techniques, including Hierarchical Navigable Small World (HNSW) graphs, Locality-Sensitive Hashing (LSH), and many others (9) could further enhance search efficiency for sparse high-dimensional datasets. Second, comprehensive evaluation on ultra-large-scale datasets ($>$ 1M points) with controlled noise levels would establish robust performance benchmarks and validate algorithmic resilience.

Additionally, four promising research directions warrant investigation: applying these optimization techniques to other clustering algorithms like spectral clustering (14); integrating compressed sensing methods to handle high-dimensional sparse data (15); developing comprehensive numerical analysis of the algorithm's stability and convergence properties (16); and deploying Chameleon2++ in complex information systems to evaluate real-world scalability (17). Extending Chameleon2++ to domain-specific applications including medical image segmentation, cybersecurity anomaly detection, and recommendation systems would help demonstrate its versatility across diverse fields.

Bibliography

[1] T. Barton, T. Bruna, and P. Kordik, "Chameleon 2: an improved graph-based clustering algorithm," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 13, no. 1, pp. 1–27, 2019.

[2] Y. Zhang, S. Ding, L. Wang, Y. Wang, and L. Ding, "Chameleon algorithm based on mutual k-nearest neighbors," *Applied Intelligence*, vol. 51, no. 4, pp. 2031–2044, 2021.

[3] Y. Zhang, S. Ding, Y. Wang, and H. Hou, "Chameleon algorithm based on improved natural neighbor graph generating sub-clusters," *Applied Intelligence*, vol. 51, no. 11, pp. 8399–8415, 2021.

[4] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.

[5] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin, "A review of clustering techniques and developments," *Neurocomputing*, vol. 267, pp. 664–681, 2017.

[6] G. Karypis, E.-H. Han, and V. Kumar, "Chameleon: Hierarchical clustering using dynamic modeling," *computer*, vol. 32, no. 8, pp. 68–75, 1999.

[7] T. Shatovska, O. Onoprienko, and A. Fedorov, "A modified multilevel approach to the dynamic hierarchical clustering for complex types of shapes," - , vol. 2, no. 11 (56), pp. 11–14, 2012.

[8] M. R. Abbasifard, B. Ghahremani, and H. Naderi, "A survey on nearest neighbor search methods," *International Journal of Computer Applications*, vol. 95, no. 25, 2014.

[9] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1475–1488, 2019.

[10] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in vlsi domain," in *Proceedings of the 34th annual Design Automation Conference*, 1997, pp. 526–529.

[11] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, "High-quality hypergraph partitioning," *ACM Journal of Experimental Algorithmics*, vol. 27, pp. 1–39, 2023.

[12] G. Karypis and V. Kumar, "A hypergraph partitioning package," *Army HPC Research Center, Department of Computer Science & Engineering, University of Minnesota*, 1998.

[13] ——, "Multilevel k-way hypergraph partitioning," in *Proceedings of the 36th annual ACM/IEEE design automation conference*, 1999, pp. 343–348.

[14] A. A. Shastri, K. Ahuja, M. B. Ratnaparkhe, A. Shah, A. Gagrani, and A. Lal, "Vector quantized spectral clustering applied to whole genome sequences of plants," *Evolutionary Bioinformatics*, vol. 15, p. 1176934319836997, 2019.

[15] R. Agrawal and K. Ahuja, "CSIS: Compressed sensing-based enhanced-embedding capacity image steganography scheme," *IET Image Processing*, vol. 15, no. 9, pp. 1909–1925, 2021.

[16] R. Choudhary and K. Ahuja, "Stability analysis of bilinear iterative rational krylov algorithm," *Linear Algebra and its Applications*, vol. 538, pp. 56–88, 2018.

[17] S. Kim, U. Murthy, K. Ahuja, S. Vasile, and E. A. Fox, “Effectiveness of implicit rating data on characterizing users in complex information systems,” in *Rauber, A and Christodoulakis, S and Tjoa, A M (eds), Research and Advanced Technology for Digital Libraries (ECDL 2005), Lecture Notes in Computer Science.* Springer, 2005, vol. 3652.