Accelerating Clustering Algorithms with Tree Data Structures

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von
Andreas Roland Lang

Dortmund
2025

Tag der mündlichen Prüfung:	29.01.2026
Dekan:	Prof. Dr. Jens Teubner
Gutachter:	Prof. Dr. Erich Schubert
	Prof. Dr. Andreas Züfle

Abstract

Clustering is a central task in unsupervised learning, enabling the discovery of structure in data without prior labels. It supports a wide range of applications, from image recognition and anomaly detection to customer analytics and text mining. Despite decades of research, classical methods such as Hierarchical Agglomerative Clustering (HAC) and k-means remain popular due to their simplicity and interpretability. Yet, scaling them to modern, large, and high-dimensional datasets is challenging.

One of the most influential approaches for scalability in this area is BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies), which introduced the Cluster Feature tree (CF-Tree) as a compact representation of data. CF-Trees summarize points using simple statistics and enable the compression of large datasets into a manageable form, suitable for efficient processing. However, BIRCH suffers from numerical instability due to numerically problematic variance computations, which can lead to catastrophic cancellation and unreliable result quality.

This thesis presents BETULA, a refinement of BIRCH that replaces unstable variance formulas with robust running-statistics computations. BETULA preserves the efficiency of CF-Trees while ensuring numerical stability, making them a reliable building block for clustering. For HAC, it enables efficient and stable approximations of common linkage methods, making exploratory analysis feasible on much larger datasets. BETULA also extends the information stored in cluster features so it can be applied to Gaussian Mixture Models (GMMs), where its (co-)variance-aware summaries allow scalable and stable optimization while providing high approximation quality. Another one of the most widely used clustering methods is k-means, which can also be accelerated with BETULA. One important aspect of k-means, critical for good result quality and runtime, is the initialization that is commonly done with k-means++. We use the variance information stored in cluster features to introduce multiple new initialization strategies (tree, trunk, leaves) based on the BETULA cluster tree that approximate k-means++ and improve initial quality and convergence speed compared to existing solutions. We also show that, when correctly applied, the BETULA approximation for k-means delivers comparable results to standard k-means algorithms while being more efficient.

For applications in which approximation is not suitable, we present Cover-means, an algorithm that accelerates k-means by integrating a Cover Tree index to prune redundant distance calculations with state-of-the-art algorithms that use geometric bounds. This non-approximative k-means algorithm achieves superior runtime performance in

a range of experimental settings and remains competitive with other approaches.

This thesis offers a detailed evaluation of clustering improvements based on BETULA and Cover-means. We compare the performance of our Cover-means approach with other state-of-the-art k-means algorithms and demonstrate how the scaling of HAC, GMM with the EM algorithm, and k-means can be improved over other methods in a numerically stable way with BETULA cluster features. Finally, we highlight the crucial role of a good initialization and its importance even for approximative techniques, directly influencing clustering quality.

Acknowledgement

The journey into accelerating clustering algorithms was challenging, and it could not have been completed without the support of many wonderful people.

To begin, I owe my deepest gratitude to Prof. Dr. Erich Schubert, my advisor, for his guidance and support whenever I needed help. His expertise, patience, and encouragement have been essential to the completion of this work. I am also thankful to my committee members Prof. Dr. Andreas Züfle, Prof. Dr. Kevin Buchin, and JProf. Dr. Thomas Liebig for their valuable advice and constructive critique. A special thanks goes to Prof. Dr. Petra Wiederkehr for her mentoring, and to Dr. Helena Kotthaus, my master's thesis supervisor, whose encouragement to pursue academic research and introduction to Erich set me on this path.

Working with my colleagues Gloria Feher, Lars Lenssen, and Erik Thordsen has been one of the most enjoyable parts of this journey. Our discussions were always inspiring and insightful, even during the difficult months when COVID-19 kept us out of our offices. I am also thankful to Roland Kühn and Christian Hakert, with whom I shared not only my first teaching experiences but also many great conversations and bike rides along the way. I am thankful to Petra Budíková and Vlastislav Dohnal for their valuable collaboration and the opportunity to extend the focus of my research.

My sincere appreciation goes to the Collaborative Research Center SFB 876 – Providing Information by Resource-Constrained Data Analysis, whose partial funding and perspective on data analysis greatly enriched my research. I am equally thankful to the Faculty of Computer Science at the Technische Universität Dortmund for their continued support and the stimulating environment they provided.

The SISAP (International Conference on Similarity Search and Applications) community deserves special mention for the many engaging discussions and genuine interest in our work. Their feedback and enthusiasm have been a real motivation and have also given us the opportunity to develop and publish extended journal versions of our research. I am also grateful to David and Camilla, whose research stays provided valuable insights into different research environments and international collaboration.

I could not have reached this point without the steady encouragement of my friends and family, who reminded me to keep going when things got difficult. Most importantly, my heartfelt thanks go to my partner, Mimka. This thesis has been a constant presence throughout most of our relationship, and beyond her help with proofreading, I am endlessly grateful for her love, patience, understanding, and support.

Contents

List of Symbols

Notation	Description
	Special Variables
$\mathcal{X}$	Database of valid data objects
$x \in \mathcal{X}$	Vector x from database $\mathcal{X}$
x_i	Component i of vector x
w_x	Weight of object x, 1 for a vector
d	Dimension of data / vectors
$\mathbb{R}^d$	The d-dimensional vector space
I	Identity matrix
	Functions
$d(x, y)$	Distance between vectors x and y
$\|x\|$	The Minkowski 1-norm of a vector x
$\|\|x\|\|$	The Minkowski 2-norm of a vector x
$\|\|x\|\|_p$	The Minkowski p-norm of a vector x
R(AB)	Radius absorption criterion for the set $AB := A \cup B$
D(AB)	Diameter absorption criterion, for the set $AB := A \cup B$ (equal to D3(A,B))
D0(A,B)	Euclidean distance criterion for sets A and B
D1(A,B)	Manhattan distance criterion for sets A and B
D2(A,B)	Inter-cluster distance criterion for sets A and B
D3(A,B)	Intra-cluster distance criterion for sets A and B
D4(A,B)	Variance-increase distance criterion for sets A and B
	Clustering
C	Cluster
k	Number of Clusters
C_1	The closest cluster
C_2	The second closest cluster
u_x	Upper bound on $d(x, c_1)$
l_x	Lower bound on $d(x, c_2)$
μ_x	Mean of entity x
r_X	Maximum distance of any entity in $max_{x \in X} d(\mu_X, x)$

Notation	Description
$a(x)$	Cluster index of object x
CF	Cluster Feature
CF^{BIRCH}	BIRCH Cluster Feature
$\text{CF}^{\text{BETULA}}$	BETULA Cluster Feature
b	Scaling factor of a Cover Tree
$\overrightarrow{LS}_X$	The component-wise sum $\sum_{x \in X} x$
SS_X	The sum of sum of squares (scalar) $\sum_i \sum_{x \in X} x_i^2$
S_X	Sum of squared deviations $\sum_{x \in X} w_x \lVert x - \mu_X \rVert^2$
	Statistics
$\text{Var}(\dots)$	Variance of a term
$E[\dots]$	Expected value of a term
π_i	Mixing weight of the i-th component
θ	Parameter set
ℓ	Log likelihood
σ^2	Variance
Σ	Covariance matrix
$\mathcal{U}$	Uniform distribution
$\mathcal{N}$	Normal distribution

List of Figures

List of Acronyms

Notation	Description
AGNES	Agglomerative Nesting
ALOI	Amsterdam Library of Object Images
ARI	Adjusted Rand Index
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies [Zha96; ZRL96; ZRL97]
BICO	BIRCH on Core Sets [Fic+13]
CF	Cluster Feature
DBSCAN	Density-Based Spatial Clustering of Applications with Noise [Est+96]
ELKI	Environment for DeveLoping KDD-Applications Supported by Index-Structures
EM	Expectation Maximization algorithm [DLR77]
GMM	Gaussian Mixture Modeling
HAC	Hierarchical Agglomerative Clustering
pdf	Probability density function
RMSD	Root Mean Squared Deviation
SAHN	Simple Agglomerative Hierarchical Nesting
SSE	Sum of Squared Errors
UPGMA	Unweighted Pair Group Method with Arithmetic mean
UPGMC	Unweighted Pair Group Method with Centroid mean
WPGMA	Weighted Pair Group Method with Arithmetic mean
WPGMC	Weighted Pair Group Method with Centroid mean

Chapter 1
Introduction

Cluster analysis is one of the central tasks in unsupervised learning. It aims to discover potential structure in a dataset when no labeled data or example patterns are available. Unlike supervised learning, there is no obvious ground truth to compare against, which makes clustering inherently underspecified and subjective [Bon64; Est02]. As a result, clustering is notoriously difficult to evaluate. In practice, it is best approached as exploratory data analysis: a way to generate hypotheses about possible structures in the data. These hypotheses can then be tested by domain-specific procedures, often requiring an expert to inspect and interpret the results manually.

Over the decades, many clustering algorithms and evaluation measures have been proposed, each with unclear advantages over the others. Well-known examples include hierarchical agglomerative clustering (HAC) [KR90] with its many linkage criteria [LW67], which computes a hierarchy of clusterings that can be used to retrieve a clustering with a given granularity or to examine different substructures in the data. Partitioning methods such as k-means [Llo57; Ste57] and k-medoids [SR21] divide the data into k groups, with k-means minimizing the sum of squared errors and k-medoids being more flexible with the metric by choosing actual data points as centers and minimizing total deviation. Probabilistic models like Gaussian Mixture Models (GMMs) take a softer approach, assigning each point a probability of belonging to a cluster rather than a hard label, and can be optimized with the EM-algorithm [DLR77]. Density-based methods such as DBSCAN [Est+96] define clusters as dense regions of points separated by sparser areas, which allows them to discover arbitrarily shaped clusters and also detect noise points. Each approach has strengths and weaknesses, but none consistently outperforms the others across all scenarios. This diversity reflects the fact that clustering is not a single well-defined task but a collection of strategies adapted to different data and application contexts.

From a computational perspective, many clustering algorithms are built on optimization problems that are NP-hard, with k-means clustering as a prominent example. For this reason, we usually rely on approximate solutions. Still, ever-increasing amounts of data raise concerns about scalability. Research has explored acceleration strategies, such as index tree structures in computer science more broadly, and their application to clustering (for example, HAC variants and DBSCAN). Even relatively fast approximations

such as k-means can be accelerated through spatial data structures like KD-trees, based on filtering by Kanungo et al. [Kan+02] and pruning by Pelleg and Moore [PM99]. These developments illustrate the constant effort to balance computational feasibility with algorithmic performance.

At the same time, clustering cannot be studied in isolation from the resources it consumes. Every clustering task demands memory, CPU cycles, and time—all of which translate into costs. Faster computation often requires more powerful (and more expensive) hardware, while resource-constrained approaches reduce costs but may delay results or sacrifice accuracy. The balance depends on the application: real-time systems may prioritize speed, whereas offline approaches can afford slower, more computationally demanding methods. The rise of edge computing adds another layer of complexity [Lap+21]. Increasingly, clustering must be performed directly on local or distributed devices, whether for privacy, latency, or bandwidth reasons. In these contexts, algorithms must be resource-aware, lightweight, and able to deliver useful results under strict computational budgets.

Beyond efficiency, there is also the issue of trust in clustering results. Part of this comes from privacy: keeping data local reduces the risk of exposure. But trust is equally tied to reliability and numerical stability. For clustering to be useful in practice, it should give reproducible results—small changes in the data or in computation should not entirely change the outcome. If an algorithm is too unstable, confidence in its results quickly disappears, especially when resource limits force the use of approximations or lower-precision calculations. In the end, effective clustering needs to strike a balance: it has to run within the available resources while still producing results that are consistent and trustworthy.

All of these challenges point to the broader theme of this thesis. Clustering is not simply a matter of uncovering patterns in data but of achieving this under practical constraints—limited time, limited resources, and the demand for reliable, interpretable outcomes. The following chapters explore how clustering algorithms can be refined and assessed with these concerns in mind.

In particular, this thesis presents BETULA, a numerically stable refinement of BIRCH, which enables reliable cluster summaries and supports efficient hierarchical and probabilistic clustering. It also presents a Cover Tree based acceleration for k-means, improving scalability by pruning redundant distance computations while preserving exactness. Both approaches are evaluated against state-of-the-art methods, showing how stability and spatial indexing can extend the applicability of classical clustering algorithms to modern large-scale data analysis.

1.1. Structural Outline

This thesis is structured into the following chapters. Chapter 2 lays out the fundamental concepts that underpin this thesis, establishing the theoretical background for the research. It covers Euclidean spaces, clustering analysis with an emphasis on hierarchical clustering, k-means, and Gaussian mixture modeling, as well as indexing techniques

using trees and the BIRCH algorithm. Together, these topics form the foundation for the proposed method, BETULA.

Chapter 3 presents my contributions to tree data structures for clustering, demonstrating how this approach can improve numerical properties. Chapter 4 introduces enhancements to hierarchical agglomerative clustering (HAC) through the BETULA algorithm. Chapter 5 explores how BETULA and the Cover Tree can improve the convergence speed of k-means clustering. Chapter 6 extends these improvements to Gaussian mixture models (GMMs), highlighting how BETULA contributes to both efficiency and stability. Finally, Chapter 7 concludes the thesis by summarizing my contributions and outlining directions for future research.

1.2. Author's Contributions and Publications

This section lists all the articles I wrote during my doctoral studies. They are presented in chronological order and divided into two categories: articles that contributed to this thesis and articles that did not. For the contributing articles, I also describe my specific role and contributions under each entry.

Articles with Contribution to This Thesis

[LS20] A. Lang and E. Schubert. "BETULA: Numerically Stable CF-Trees for BIRCH Clustering". In: *Similarity Search and Applications - 13th International Conference, SISAP 2020, Copenhagen, Denmark, September 30 - October 2, 2020, Proceedings*. Vol. 12440. Lecture Notes in Computer Science. Cham: Springer, 2020, pp. 281–296. ISBN: 978-3-030-60936-8. DOI: 10.1007/978-3-030-60936-8_22

My contribution: Conceptualization, implementation, experiments, writeup. Erich Schubert contributed to the conceptualization and writeup.

[LS21] A. Lang and E. Schubert. "BETULA: Fast clustering of large data with improved BIRCH CF-Trees". In: *Information Systems* 108 (2021), p. 101918. ISSN: 0306-4379. DOI: 10.1016/J.IS.2021.101918

My contribution: Conceptualization, implementation, experiments, writeup. Erich Schubert contributed to the conceptualization and writeup.

[SL22] E. Schubert and A. Lang. "Data Aggregation for Hierarchical Clustering". In: *Machine Learning under Resource Constraints - Volume 1: Fundamentals*. De Gruyter STEM. De Gruyter, 2022, pp. 215–226. DOI: 10.1515/9783110785944-005

My contribution: Conceptualization, implementation, experiments, writeup. Erich Schubert contributed to the conceptualization and writeup.

[LS23] A. Lang and E. Schubert. "Accelerating k-Means Clustering with Cover Trees". In: *Similarity Search and Applications - 16th International Conference, SISAP 2023, A Coruña, Spain, October 9-11, 2023, Proceedings*. Vol. 14289. Lecture Notes in Computer Science. Springer, 2023, pp. 148–162. DOI: 10.1007/978-3-031-46994-7_13

My contribution: Conceptualization, implementation, experiments, writeup. Erich Schubert contributed to the conceptualization and writeup.

Articles with no Contribution to This Thesis

[SLF21] E. Schubert, A. Lang, and G. Feher. "Accelerating Spherical k-Means". In: *Similarity Search and Applications - 14th International Conference, SISAP 2021, Dortmund, Germany, September 29 - October 1, 2021, Proceedings*. Vol. 13058. Lecture Notes in Computer Science. Springer, 2021, pp. 217–231. DOI: 10.1007/978-3-030-89657-7_17

[Ján+24] M. Jánošová, A. Lang, P. Budíková, E. Schubert, and V. Dohnal. "Advancing the PAM Algorithm to Semi-supervised k-Me-doids Clustering". In: *Similarity Search and Applications - 17th International Conference, SISAP 2024, Providence, RI, USA, November 4-6, 2024, Proceedings*. Vol. 15268. Lecture Notes in Computer Science. Springer, 2024, pp. 223–237. DOI: 10.1007/978-3-031-75823-2_19

[Ján+26] M. Jánošová, A. Lang, P. Budikova, E. Schubert, and V. Dohnal. "On the evaluation and optimization of LabeledPAM". in: *Information Systems* 135 (2026), p. 102580. ISSN: 0306-4379. DOI: 10.1016/j.is.2025.102580

1.3. Use of AI-Assisted Tools

For the writing of this thesis, I used AI-assisted tools like Grammarly and languagetool.org for grammar and style improvements only. Their suggestions were reviewed and thoroughly checked for factual errors. I claim full responsibility for the content of this thesis.

Chapter 2
Preliminaries

This chapter introduces the theoretical foundations and essential terminology required for the developments presented in the later parts of this thesis. Since clustering methods are rooted in concepts from linear algebra, probability theory, and computational geometry, it is important to establish a clear and consistent foundation before discussing novel contributions. The material covered here provides the mathematical and algorithmic basis upon which the proposed methods are built. It is based on the foundations of Lang and Schubert [LS20; LS21; LS23] and Schubert and Lang [SL22] and extended with further information.

2.1. Euclidean Space

Most of the work in this thesis focuses on points within Euclidean vector space [BCL09]. We start by recalling the definition of the d-dimensional Euclidean space, denoted $\mathbb{R}^d$. This space consists of all ordered d-tuples of real numbers and serves as a fundamental structure in mathematics. It is equipped with an inner product (dot product) that induces a norm and metric, enabling geometric interpretations in computational contexts. Formally, for vectors $x, y \in \mathbb{R}^d$, the dot product is defined as:

$$x \cdot y = \sum_{i=1}^{d} x_i y_i.$$

Through this inner product, one can define the Euclidean norm:

$$\|x\|_2 = \sqrt{x \cdot x} = \sqrt{\sum_{i=1}^{d} x_i^2}.$$

The associated metric, or *Euclidean distance* between vectors x and y, is then given by:

$$d(x, y) = \|x - y\|_2 = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}. \tag{2.1}$$

This metric computes the straight-line distance between data points, rooted in Euclidean geometry. Its intuitive interpretation and computational simplicity make it one of the most widely used measures in computer science, particularly in low-dimensional settings. Many machine learning algorithms make use of it, including k-means [Llo57], BIRCH [ZRL96] and KD-Tree [Ben75]. Euclidean structures also support dimensionality reduction techniques like Principal Component Analysis (PCA) [Jol86]. Often data that is not by nature Euclidean is embedded into Euclidean space, e.g., face recognition data [SKP15]. However, while Euclidean distance remains efficient for small dimensions, its performance often deteriorates in high-dimensional spaces due to the "curse of dimensionality" [AHK01].

2.1.1. Triangle Inequality

The triangle inequality, cf. Zezula et al. [Zez+06], is a fundamental property of Euclidean spaces. For any three objects $x, y, z \in \mathbb{R}^d$ with a valid metric $d(\cdot,\cdot)$, it holds that

$$d(x,z) \leq d(x,y) + d(y,z).$$

This reflects the geometric principle that the straight-line path between two points is shortest, implying no side of a triangle can exceed the sum of the other two sides. For a non-degenerate Euclidean triangle with side lengths a, b, and c, this yields:

$$\begin{aligned} a + b &> c, \\ a + c &> b, \\ b + c &> a. \end{aligned}$$

Strict inequalities hold when the triangle has non-zero area. This property is essential in computational geometry, clustering algorithms (e.g., k-means), and metric learning, where distance metrics must satisfy this condition for theoretical guarantees. Schubert [Sch21a] derives a similar inequality for cosine distance.

2.2. Cluster Analysis

Cluster analysis is a fundamental technique in data mining that involves grouping similar objects into clusters while keeping dissimilar objects separate. This unsupervised learning approach reveals hidden patterns and structures within datasets without requiring prior knowledge of class labels. By identifying natural groupings, cluster analysis enables applications ranging from market segmentation to image compression and customer preference identification.

Cluster analysis necessitates an iterative process that requires researcher involvement at multiple stages—including variable selection, similarity measure choice, and subset focus—rather than a single mechanical application [Eve+11]. Because of the iterative nature of cluster analysis, scalability is often of major concern [ZRL96]. There exist many different clustering algorithms, each with its own strengths and weaknesses.

2.3. Hierarchical Agglomerative Clustering

Among the various clustering approaches, hierarchical methods are some of the oldest but still widely used clustering methods. They can broadly be split into two main categories: Divisive Hierarchical Clustering and Hierarchical Agglomerative Clustering (HAC) [MC12]. Both approaches are particularly well-suited for problems that exhibit a hierarchical structure.

In this work we focus on HAC because it integrates very well with the BIRCH algorithm [ZRL96]. The procedure begins by treating each data point as an individual cluster. At each step, the two clusters with the smallest distance are merged, and this process is repeated until all data points are combined into a single cluster. This straightforward idea is also reflected in an alternative name for HAC – Simple Agglomerative Hierarchical Nesting (SAHN). Hierarchical clustering can usually be combined with different distance functions as well as with different inter-cluster distance definitions (commonly referred to as linkages).

2.3.1. Hierarchical Clustering Linkages

Because HAC is based on repeatedly merging the two closest clusters, it is necessary to define an appropriate distance measure between clusters, not just between individual objects. Ideally, this distance should be consistent with the underlying pairwise distance. This notion of "cluster distance" is commonly referred to as the linkage criterion, and its choice strongly influences the shape and structure of the resulting clusters as well as the algorithms that can be applied.

The two most widely used linkage strategies are single linkage and complete linkage, where the distance between two clusters is defined as the minimum or maximum distance between any pair of points, respectively. Many linkage criteria have been proposed, with several dating back to the 1950s (e.g., McQuitty [McQ57], Sneath [Sne57], Sokal and Sneath [SS63], and Wishart [Wis69]). More recent proposals include Mini-Max linkage introduced by Ao et al. [Ao+05] and Medoid-based linkages proposed by Herr et al. [Her+16], Miyamoto, Kaizu, and Endo [MKE16], and Schubert [Sch21b].

Several (though not all) linkage strategies can be expressed in terms of the recurrences introduced by Lance and Williams [LW66; LW67], which provides computational advantages for updating distances efficiently after each merge. In fact, WPGMA (McQuitty) and WPGMC (Median linkage, Sneath) can only be defined in recursive form and cannot be expressed in closed form based solely on the sets of points.

The Lance–Williams formula is given by:

$$d(A \cup B, C) = \alpha_A d(A, C) + \alpha_B d(B, C) + \beta d(A, B) + \gamma |d(A, C) - d(B, C)|, \tag{2.2}$$

where different linkage strategies correspond to different choices of the coefficients α_A, α_B, β, and γ (see Tab. 2.1). These coefficients may depend on the sizes of the clusters A, B, and C, denoted by n_A, n_B, and n_C. For brevity, we use the shorthand $n_{AB} := n_{A \cup B} = n_A + n_B$ and $n_{ABC} := n_{A \cup B \cup C} = n_A + n_B + n_C$.

Table 2.1.: Common linkages in terms of Lance-Williams factors.

Linkage	α_A	α_B	β	γ	Init.
Single	$1/2$	$1/2$	0	$-1/2$	$d(a,b)$
Complete	$1/2$	$1/2$	0	$1/2$	$d(a,b)$
Group-average (UPGMA)	$\frac{n_A}{n_{AB}}$	$\frac{n_B}{n_{AB}}$	0	0	$d(a,b)$
McQuitty (WPGMA)	$1/2$	$1/2$	0	0	$d(a,b)$
Centroid (UPGMC)	$\frac{n_A}{n_{AB}}$	$\frac{n_B}{n_{AB}}$	$-\frac{n_A \cdot n_B}{{n_{AB}}^2}$	0	$d(a,b)^2$
Median (WPGMC)	$1/2$	$1/2$	$-1/4$	0	$d(a,b)^2$
Ward	$\frac{n_{AC}}{n_{ABC}}$	$\frac{n_{BC}}{n_{ABC}}$	$-\frac{n_C}{n_{ABC}}$	0	$d(a,b)^2$

An additional and often overlooked detail concerns the initialization of the distance matrix. While single-link, complete-link, and group-average linkage work with any distance measure, Centroid linkage clustering, Ward linkage clustering, and Median linkage clustering require initialization with squared distances. These methods are inherently tied to the Euclidean distance and variance, and failure to account for this distinction (including interpretation of the results) can lead to incorrect outcomes according to Murtagh and Legendre [ML14]. The reason becomes evident when considering the respective objective functions of these methods or their closed-form formulations (cf. Eqs. 2.3–2.5).

Consider **single-linkage** first (and, by substituting max for min, complete-linkage). Here, the aim is to merge clusters A and B with the smallest distance between their points, i.e., with the smallest $d_{\text{single}}(A,C) := \min_{a\in A, c\in C} d(a,c)$. If both clusters consist of a single element, we obviously have $d_{\text{single}}(\{a\},\{c\}) = d(a,c)$, and we can recursively compute this linkage using $d_{\text{single}}(A \cup B, C) = \min\{d_{\text{single}}(A,C), d_{\text{single}}(B,C)\}$. The correspondence to the weights in Tab. 2.1 follows from the elementary identity $\min(x,y) = \frac{1}{2}(x+y) - \frac{1}{2}|x-y|$. Substituting $x = d_{\text{single}}(A,C)$ and $y = d_{\text{single}}(B,C)$ yields

$$\begin{aligned} d_{\text{single}}(A \cup B, C) &= \min(d_{\text{single}}(A,C),\, d_{\text{single}}(B,C)) \\ &= \tfrac{1}{2}\, d_{\text{single}}(A,C) + \tfrac{1}{2}\, d_{\text{single}}(B,C) - \tfrac{1}{2}\, |d_{\text{single}}(A,C) - d_{\text{single}}(B,C)|, \end{aligned}$$

which is exactly the Lance–Williams formula 2.2 with $\alpha_A = \alpha_B = \frac{1}{2}$, $\beta = 0$, and $\gamma = -\frac{1}{2}$. Replacing $-\frac{1}{2}$ with $+\frac{1}{2}$ switches min to max, giving complete linkage.

Group-average linkage, also known as Unweighted Pair Group Method with Arithmetic mean (UPGMA), is another intuitive linkage, and is often considered one of the best to use in practice. The idea is to capture the average distance between elements from

different clusters, i.e., $d_{\text{avg}}(A,C) := \frac{1}{n_A n_C}\sum_{a\in A}\sum_{c\in C} d(a,c)$. Clearly, for one-elemental clusters, we have $d_{\text{avg}}(\{a\},\{c\}) = d(a,c)$. The recursive formula is easy to derive:

$$\begin{aligned} d_{\text{avg}}(A\cup B, C) =& \frac{1}{n_{AB}n_C}\Big(\sum_{a\in A}\sum_{c\in C} d(a,c) + \sum_{b\in B}\sum_{c\in C} d(b,c)\Big) \\ =& \frac{1}{n_{AB}n_C}\left(n_A n_C d_{\text{avg}}(A,C) + n_B n_C d_{\text{avg}}(B,C)\right) \\ =& \frac{n_A}{n_{AB}} d_{\text{avg}}(A,C) + \frac{n_B}{n_{AB}} d_{\text{avg}}(B,C) \ . \end{aligned} \tag{2.3}$$

The term **"weighted"** (introduced by Sokal and Sneath [Sne57; SS63]) can be confusing, as it refers to the influence each point has. In "unweighted" group average, each *object* has the same weight (and, hence, the weight of each cluster is proportional to the number of objects contained), whereas in the "weighted" version i.e McQuitty and Median linkage, each *cluster* has the same weight (and, hence, each object in a larger cluster has a reduced weight). As easily seen in Tab. 2.1, both "weighted" versions correspond to their "unweighted" counterparts if we fix the cluster sizes to a constant $n_A = n_B := 1$, i.e., ignoring the cluster sizes when merging.

Among these weighted variants, the **McQuitty's** Weighted Pair-Group Method with Arithmetic mean (WPGMA [McQ57]) can be recursively defined as

$$d_{\text{McQ}}(A\cup B, C) = \tfrac{1}{2}(d_{\text{McQ}}(A,C) + d_{\text{McQ}}(B,C)),$$

which introduces an unfortunate dependency on the "merge history" of the child clusters A and B. Given three objects a, b, c, merging a and b first, then with c may yield a different result than first merging one of the other pairs. A similar argument holds for median linkage (WPGMC), discussed below.

Unweighted Pair-Group Method using Centroids (UPGMC), also known as **centroid linkage** combines clusters by the distance of the cluster means $\mu_A = \frac{1}{|A|}\sum_{x\in A} x$, i.e., always merges the smallest $d_{\text{cent}}(A,C) = \|\mu_A - \mu_C\|$. Computing the distances between the means explicitly requires many additional distance computations, and therefore is slower and less resource-efficient than a recurrent approach. But there is a special relationship between the mean, the variance, and the squared Euclidean distance that we can exploit to compute this special case elegantly with a recurrence. We discuss this relationship, without loss of generality, only for univariate data, because the squared Euclidean is simply the sum of the squared variates. We then have $\|\mu_A - \mu_C\|^2 = \mu_A^2 + \mu_C^2 - 2\mu_A\mu_C$ and obtain

$$\begin{aligned} d_{\text{cent}}(A\cup B, C) =& \frac{n_A}{n_{AB}} d_{\text{cent}}(A,C) + \frac{n_B}{n_{AB}} d_{\text{cent}}(B,C) - \frac{n_A n_B}{{n_{AB}}^2} d_{\text{cent}}(A,B) \\ =& \mu_{AB}^2 + \mu_C^2 - 2\mu_{AB}\mu_C = \|\mu_{AB} - \mu_C\|^2 \ . \end{aligned} \tag{2.4}$$

This means that for squared Euclidean distances, we can compute the distance of the means without computing the means themselves. Hence, we need to initialize the distance matrix with squared Euclidean distances, and also need to interpret the resulting linkage distances as such squared values.

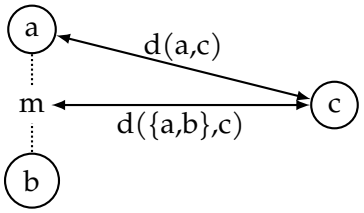


Figure 2.1.: Example why Median and Centroid linkages are non-monotone: the midpoint m of the merged cluster $\{a, b\}$ is closer to c than any of its clusters members a and b were.

The idea of **median linkage** (often referred to as Weighted Pair-Group Method using Centroids, WPGMC) is to minimize the distance of the medians, $\|m_{A\cup B} - m_C\|$, where the median is recursively defined as $m_{A\cup B} = \frac{1}{2}(m_A + m_B)$, the midpoint of the previous medians. For squared Euclidean distances, we again have a recurrent formula; the derivation is exactly as for centroid linkage, but with fixed $n_A = n_B = 1$. Median linkage and centroid linkage have the oddity that the distance $d(A \cup B, C)$ can be less than the distance of $d(A, C)$, which can yield non-monotone dendrograms: if we draw a tree representing the cluster merges, and use the linkage distance as the height of a branch, the resulting tree does not monotonously grow. These anomalies, known as inversions, arise exclusively when a linkage lacks the reducibility property defined by Bruynooghe [Bru77]. Intuitively, this happens when the new center is between two well-separated clusters, and then closer to a third than either of the two, as illustrated in Fig. 2.1.

The popular **Ward linkage** optimizes the criterion [And73; KR90; Wis69]:

$$d_{\text{Ward}}(A, B) = \frac{2n_A \cdot n_B}{n_{AB}} \|\mu_A - \mu_B\|^2 \quad . \tag{2.5}$$

The factor 2 in this equation ensures that $d_{\text{Ward}}(\{a\}, \{b\}) = \|a, b\|^2$, as desired for one-elemental clusters. This criterion can be described as "minimum increase in the sum of squares" [Pod89], which may come as a surprise given that the equation only uses the means, and does not appear to contain the sum of squares. The reader may have noticed that k-means clustering also minimizes the sum of squares. The main difference here is that Ward linkage imposes a hierarchical structure on the result, whereas k-means imposes a flat partitioning into k partitions. Usually, the result of Ward linkage cut at k partitions will be (often substantially) worse than that of k-means (for the consistency reasons explained in Schubert [Sch21b] for the case of medoid linkage), but on the other hand, k-means results for varying k will usually not nest into a hierarchy of clusters. Equation 2.5 can be obtained from rewriting the increase in the sum of squares via the König-Huygens theorem:

$$\begin{aligned} d_{\text{Ward}}(A, B) &= \sum_{x\in A\cup B} \|x - \mu_{AB}\|^2 - \sum_{a\in A} \|a - \mu_A\|^2 - \sum_{b\in B} \|b - \mu_B\|^2 \\ &= \frac{2n_A n_B}{n_{AB}} \|\mu_A - \mu_B\|^2 . \end{aligned}$$

The Lance-Williams recurrence in Table 2.1 follows by expanding the squared norm. Pulling a factor of $\frac{1}{n_{AB}}$ inside removes the denominators ($\mu_{AB}-\mu_C = \frac{1}{n_{AB}}(n_A(\mu_A-\mu_C) + n_B(\mu_B-\mu_C))$):

$$\begin{aligned} d_{\text{Ward}}(A\cup B, C) =& \frac{2n_{AB}n_C}{n_{ABC}} \|\mu_{AB}-\mu_C\|^2 \\ =& \frac{2n_C}{n_{ABC}n_{AB}} \|n_A(\mu_A-\mu_C) + n_B(\mu_B-\mu_C)\|^2 \\ =& \frac{2n_C}{n_{ABC}n_{AB}} (n_A^2 \|\mu_A-\mu_C\|^2 + n_B^2 \|\mu_B-\mu_C\|^2 + 2n_An_B(\mu_A-\mu_C)(\mu_B-\mu_C)) \end{aligned}$$

Replacing $2(\mu_A-\mu_C)(\mu_B-\mu_C) = \|\mu_A-\mu_C\|^2 + \|\mu_B-\mu_C\|^2 - \|\mu_A-\mu_B\|^2$ and collecting:

$$\begin{aligned} =& \frac{2n_C}{n_{ABC}n_{AB}} (n_An_{AB} \|\mu_A-\mu_C\|^2 + n_Bn_{AB} \|\mu_B-\mu_C\|^2 - n_An_B \|\mu_A-\mu_B\|^2) \\ =& \frac{2n_An_C}{n_{ABC}} \|\mu_A-\mu_C\|^2 + \frac{2n_Bn_C}{n_{ABC}} \|\mu_B-\mu_C\|^2 - \frac{2n_An_Bn_C}{n_{AB}n_{ABC}} \|\mu_A-\mu_B\|^2 \end{aligned}$$

Substituting $\|\mu_X-\mu_Y\|^2 = \frac{n_X+n_Y}{2n_Xn_Y} d_{\text{Ward}}(X, Y)$ from 2.5:

$$= \frac{n_{AC}}{n_{ABC}} d_{\text{Ward}}(A, C) + \frac{n_{BC}}{n_{ABC}} d_{\text{Ward}}(B, C) - \frac{n_C}{n_{ABC}} d_{\text{Ward}}(A, B).$$

2.3.2. AGNES

The standard HAC algorithm, known as AGglomerative NESting AGNES [KR90], requires storing all pairwise distances in a distance matrix. When two clusters are merged, the corresponding rows and columns of this matrix are updated according to the Lance-Williams equations. AGNES can be combined with different distance functions and various inter-cluster distance definitions (commonly referred to as linkages). This bottom-up approach produces a hierarchical cluster structure, typically visualized as a dendrogram.

The algorithm proceeds as follows:

1. Initialize n singleton clusters: $C_i = \{x_i\}$ for each data point $x_i \in \mathcal{X}$.
2. Compute the distance matrix D of size $n\times n$, where elements represent inter-cluster distances. The choice of linkage criterion determines how distances are computed between clusters.
3. Find the pair of clusters with minimal distance: $\arg\min_{i,j} d(C_i, C_j)$. Let us denote them C_p and C_q.
4. Merge C_p and C_q into a new cluster $C_k = C_p \cup C_q$, reducing the total cluster count by one.
5. Update the distance matrix to include C_k and remove C_p and C_q using the selected linkage criterion, e.g., with the Lance-Williams equations.
6. Repeat steps 3–5 until all points form a single cluster ($n-1$ merges).

The algorithm terminates when only one cluster remains, producing a dendrogram in which horizontal lines represent merge events, and the vertical axis indicates the corresponding merge distance.

2.3.3. Anderberg's Algorithm

An early approach, by Anderberg [And73] proposes an optimization technique for agglomerative hierarchical clustering that significantly reduces practical computational overhead. The method is initialized by identifying and caching the minimum dissimilarity value in each row of the distance matrix alongside its column index, requiring $O(n^2)$ time.

During iterative cluster merges, the next merge candidate is retrieved from these cached minima in constant time ($O(1)$). Crucially, upon merging clusters C_i and C_j, updates to the distance matrix are handled through targeted operations:

- if a newly computed dissimilarity improves the cached minimum for a cluster, it is updated in $O(1)$;
- when one of the merged clusters was the source of a cached minimum, a linear scan ($O(n)$) recalculates the row minimum;
- the merged cluster's dissimilarities always undergo an $O(n)$ scan to establish new minima.

Although individual merge operations retain a worst-case complexity of $O(n^2)$—and thus the theoretical upper bound for the full algorithm remains $O(n^3)$—empirical analysis suggests that row recomputations occur infrequently in practice. Assuming an average of only one row scan per merge step, the expected runtime complexity of agglomerative nesting (AGNES) reduces from cubic to quadratic $O(n^2)$. This distinction between worst-case theoretical bounds and practical performance underscores how algorithmic optimizations can yield substantial efficiency gains in real-world applications despite unaltered asymptotic limits.

2.3.4. NN-Chain

Beyond matrix-based formulations such as AGNES and Anderberg's stored-matrix approach, a different strategy employs Nearest-Neighbor chains (NN-chains) [Mur83]. An NN-chain is a sequence of points where each point is the nearest neighbor of the one before it. The sequence starts with an arbitrary point i. From there, the next point is its nearest neighbor $j = \mathrm{NN}(i)$, followed by $k = \mathrm{NN}(j)$, and so on. This continues until the chain reaches a pair of points, p and q, that are nearest neighbors of each other, meaning $p = \mathrm{NN}(q)$ and $q = \mathrm{NN}(p)$. The chain always terminates with a reciprocal nearest-neighbor pair, and its structure guarantees monotonically decreasing dissimilarities without cycles.

Instead of maintaining or repeatedly updating a full distance matrix, the algorithm incrementally constructs chains, merging clusters whenever a reciprocal pair is found.

This yields significant computational savings, particularly for single linkage and complete linkage, where NN-chains achieve an $O(n^2)$ complexity compared to the $O(n^3)$ cost of naive HAC implementations [Mur83]. This property makes them especially attractive for hierarchical clustering on large or high-dimensional datasets. For linkages that satisfy the reducibility property defined by Bruynooghe [Bru77] the results are identical to the stored matrix approaches.

Modern fast HAC algorithms extend this idea by combining NN-chains with advanced data structures, like Schubert [Sch24] who uses an "incremental priority search" index to reduce computational overhead even further.

2.4. *k*-Means

In contrast to hierarchical approaches, partitioning-based methods aim to divide the data directly into a fixed number of clusters. One of the most prominent representatives is *k*-means, a simple yet powerful algorithm that has become the de facto standard in practice (commonly attributed to Lloyd [Llo57], though earlier variants were described by Steinhaus [Ste57] and Forgy [For65]; cf. Bock [Boc07]).

In *k*-means, the data is approximated using k centers, which are the arithmetic mean of the partitions, and the goal is to minimize the sum of squared deviations of all samples and their nearest centroids. Finding the true optimum is NP-hard, as shown by Mahajan, Nimbhorkar, and Varadarajan [MNV09], which motivates heuristic approaches such as the standard algorithm. The popularity of *k*-means and an ever-increasing amount of data led to many improvements to the standard algorithm. Many replicate the convergence behavior of the standard algorithm exactly and are thus termed "exact" *k*-means, in contrast to methods that approximate one of the two update steps. The runtime is dominated by distance computations between objects and cluster centers in each iteration. One line of acceleration is the approximation of the data itself, e.g., by sampling [BFR98] or aggregation, which is used in mini-batch *k*-means [Scu10], BICO [Fic+13], and BETULA [LS21], among others. These approaches usually yield results close to the standard algorithm, since the means in *k*-means are statistical summaries themselves, and in some cases guarantees on error bounds are available [Fic+13].

Algorithms that are replicating the "exact" *k*-means algorithm can be split further into two groups. The first group uses data structures like kd-trees are used to partition the data space recursively, allowing subsets of points to be assigned simultaneously via distance bounds between centers and bounding boxes of tree nodes, as in Kanungo et al. [Kan+02] and Pelleg and Moore [PM99]. Second, methods based on the prune redundant distance computations by exploiting the fact that many points remain in the same cluster after a few iterations. Phillips [Phi02] used pairwise distances between centers for pruning. Elkan [Elk03] extended this idea to maintaining upper and lower bounds for each point-center distance. This requires $O(kn)$ storage and considerable update effort as centers move. Hamerly [Ham10] reduced memory costs by merging all lower bounds to distant centers into a single bound, at the cost of additional distance computations due to looser bounds. Yingyang *k*-means [Din+15] strikes a balance by using only

$t < k$ bounds. State-of-the-art methods such as Exponion [NF16] and Shallot [Bor20] extend this idea by considering (hyper-)balls around the closest, and in Shallot also the second-closest, centers to further reduce candidate distances.

No single method dominates in all cases; efficiency depends on dimensionality, size of dataset, number of clusters, initialization, and data distribution. Because pruning-based methods must compute all distances in the first iteration to establish bounds, their initial cost equals that of the standard algorithm. The advantages arise only in later iterations, when cluster centers move little. Recent work by Yu, Chen, and Chen [YCC20] includes leveraging distances to previous center locations to improve pruning, and by Schubert [Sch21a] and Schubert, Lang, and Feher [SLF21], extending these techniques to spherical k-means for text clustering, where cosine similarity replaces Euclidean distance and a corresponding triangle inequality is applied.

Even the "exact" k-means algorithm is only guaranteed to find a local minimum for the sum of squared errors, so the initialization of the cluster centers strongly influences both runtime and solution quality. Arthur and Vassilvitskii [AV07] introduced the k-means++ initialization that has become the most popular choice, as it already provides a (probabilistic) quality guarantee by sampling centers proportionally to their expected contribution to variance reduction. Even with the k-means++ initialization, it is common practice to do repeated runs and take the best result.

2.4.1. Standard k-Means Algorithm

The standard k-means algorithm is a heuristic for partitioning points $x \in \mathcal{X}$ into k clusters $\{C_1, ..., C_k\}$. It begins with the initial cluster centers $c_1, ..., c_k$ (e.g., sampled from the data) and alternates between the following two optimization steps:

1. Each object x is assigned to its nearest cluster center, using the squared Euclidean distance, as defined in Eq. (2.6), where $a(x)$ denotes the cluster index:

$$a(x) := \underset{i \in 1,..,k}{\arg\min} \|x - c_i\|_2^2, x \in X. \tag{2.6}$$

2. Each cluster center c_i is recomputed as the mean of its assigned points using:

$$c_i := \frac{\sum_{x|a(x)=i} x}{|\{x \mid a(x) = i\}|} \quad i \in 1, ..., k. \tag{2.7}$$

The algorithm terminates when no assignments change. Alternatives exist where early termination is possible when the movement of cluster centers falls below a predefined threshold or a set number of iterations is reached. Although this heuristic does not generally find the global optimum, Selim and Ismail [SI84] show that it converges to a local fixed point when no assignment changes.

2.4.2. Triangle Inequality in k-Means

Many accelerated k-means algorithms exploit the triangle inequality to reduce the number of distance calculations in Eq. 2.6 to all centers. This idea was first introduced by Phillips [Phi02]. Given an object x and a cluster center c_i with known distance $d(x, c_i)$, and some other center c_j with an unknown distance, the triangle inequality yields:

$$d(c_i, c_j) \leq d(x, c_i) + d(x, c_j) \tag{2.8}$$

$$\Rightarrow \quad d(x, c_j) \geq d(c_i, c_j) - d(x, c_i). \tag{2.9}$$

This provides a lower bound on distance $d(x, c_j)$, and can sometimes eliminate the need to compute it explicitly. In particular, Eq. Eq. 2.9 [Phi02] implies :

$$d(c_i, c_j) \geq 2d(x, c_i) \quad \Rightarrow \quad d(x, c_j) \geq d(x, c_i) \ . \tag{2.10}$$

Hence, any center sufficiently far from the current closest center can be excluded with this inexpensive filter. Since all pairwise distances $d(c_i, c_j)$ can be computed once per iteration, the overhead remains manageable as long as k is not too large.

Another application for the triangle inequality in k-means is the use of per-object bounds – an upper bound on the distance to the assigned cluster and lower bounds on the distances to all other clusters:

$$u_x \geq d(x, c_{a(x)}) \qquad \text{and} \qquad l_x \leq d(x, c_j) \qquad \forall j \neq a(x).$$

Then $u_x \leq l_x$ implies that $a(x)$ is still the closest cluster by

$$d(x, c_{a(x)}) \leq u_x \leq l_x \leq d(x, c_j).$$

In particular, these bounds were introduced by Elkan [Elk03] and later simplified by Hamerly [Ham10], who used a single lower bound for all other clusters.

To ensure correctness, the bounds must be updated whenever centers move. This strategy becomes more effective once cluster centers shift only slightly. While centers still move substantially, $u_x > l_x$ occurs frequently, forcing many true distance computations. In Hamerly's method, even a single cluster moving far can trigger widespread recomputations. By contrast, Elkan's version requires fewer distance evaluations, but has to update $N \cdot k$ bounds each iteration. The current state-of-the-art algorithms, such as Exponion, introduced by Newling and Fleuret [NF16] and Shallot, introduced by Borgelt [Bor20], adopt Hamerly-style bounds as their foundation as well.

When the distances of an object x to all cluster means c_i are calculated, the upper bound u_x is set to the distance to the assigned cluster $a(x)$ and the lower bound l_x to the minimum distance to any of the other clusters c_j for $j \neq a(x)$:

$$u_x := d(x, c_{a(x)}) \qquad\qquad l_x := \min_{j \neq a(x)} d(x, c_j).$$

In each iteration, after updating the centers, the bounds are adjusted based on how

far the means have moved. For each mean c_i, we compute how far it moved from its previous location c'_i. Then, the distance moved by the nearest mean is added to the upper bound and the maximum distance moved by any other cluster mean from the lower bound is subtracted:

$$u_x := u_x + d(c'_{a(x)}, c_{a(x)}) \qquad\qquad l_x := l_x - \max_{j \neq a(x)} d(c'_j, c_j).$$

A further refinement of Hamerly's approach, the Exponion [NF16] algorithm, introduces spherical bounds to limit the number of candidate centers considered for each object. Instead of comparing against all cluster centers, Exponion checks only those inside a hypersphere around the currently assigned center. This greatly reduces the number of distance computations, especially in low- and medium-dimensional spaces, while maintaining the correctness guarantees of Hamerly's algorithm.

Building on this idea, Shallot [Bor20] further improves pruning by introducing a second sphere around the second-closest center. By maintaining both the closest and second-closest candidate regions, Shallot achieves tighter bounds and reduces the number of clusters that must be evaluated per object.

Together, these refinements demonstrate how the triangle inequality remains central to the design of efficient k-means algorithms. By combining global bounds (Elkan, Hamerly) with geometric constraints (Exponion, Shallot), modern methods achieve substantial speedups while preserving exact convergence behavior.

2.5. Gaussian Mixture Modeling

Gaussian Mixture Models (GMMs) constitute a probabilistic framework for representing data generated from multiple latent Gaussian distributions. This approach assumes that each observation is drawn from one of K components, with the mixture providing a soft assignment to clusters through posterior probabilities. GMMs offer significant flexibility in various data distributions that cannot be captured by single Gaussian models.

2.5.1. Mathematical Formulation

A Gaussian Mixture Model defines the probability density of a d-dimensional data object $x \in \mathcal{X}$ as a weighted sum of k Gaussian components:

$$p(x \mid \theta) = \sum_{i=1}^{k} \pi_i \mathcal{N}(x \mid \mu_i, \Sigma_i),$$

where $\pi_i > 0$ denotes the mixing weight of the i-th component ($\sum_{i=1}^{k} \pi_i = 1$), and $\mathcal{N}(x \mid \mu_i, \Sigma_i)$ is a multivariate Gaussian density with mean vector μ_i and covariance matrix Σ_i. The full parameter set is $\theta = \{\pi_i, \mu_i, \Sigma_i\}_{i=1}^{k}$.

The multivariate Gaussian density for component i is explicitly given by

$$\mathcal{N}(x \mid \mu_i, \Sigma_i) = \frac{1}{(2\pi)^{d/2} |\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)\right), \tag{2.11}$$

where the quadratic form $(x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)$ represents the squared Mahalanobis distance between x and μ_i. This metric accounts for feature correlations, generalizing Euclidean distance to anisotropic clusters.

Given a dataset $\mathcal{X} = \{x_1, \ldots, x_N\}$, our goal is to find the parameters θ that best explain this observed data. We quantify this using the likelihood function, $L(\theta \mid \mathcal{X})$, which is defined as the probability of observing the dataset $\mathcal{X}$ given a particular set of parameters θ, denoted as $p(\mathcal{X} \mid \theta)$.

Under the common assumption that the data points $\{x_1, \ldots, x_N\}$ are independent and identically distributed (i.i.d.), the joint probability of the entire dataset can be calculated as the product of the probabilities of each individual data point. Therefore, the likelihood function is given by:

$$L(\theta \mid \mathcal{X}) = \prod_{i=1}^{N} p(\mathbf{x}_i \mid \theta), \tag{2.12}$$

and the corresponding log-likelihood, which improves numerical stability and simplifies optimization, is

$$\ell(\theta \mid \mathcal{X}) = \sum_{i=1}^{N} \log\left(\sum_{j=1}^{k} \pi_j \mathcal{N}(\mathbf{x}_i \mid \mu_j, \Sigma_j)\right). \tag{2.13}$$

2.5.2. Covariance Structures and Model Complexity Trade-offs

The structure of Σ_i critically influences model expressiveness, parameter count, and computational cost. Four common variants are distinguished based on their covariance assumptions:

Spherical (or Isotropic) Gaussians One of the simplest cases is the spherical model, in which each component is assumed to have a covariance matrix of the form $\Sigma_k = \sigma_i^2 I$, where I is the identity matrix and σ_i^2 is the variance parameter of the i-th component. Under this assumption, the variance is identical in all directions, which means that the contours of equal probability density are spheres (or circles in two dimensions). Each component is therefore parameterized by a single variance term σ_i^2. This formulation is appropriate when the underlying clusters are roughly spherical and exhibit similar spread across all dimensions. The main advantage of this model is its simplicity: it has very low computational cost, requires fewer parameters, and generally converges quickly.

Diagonal Covariance Gaussians A more generic alternative is the diagonal covariance model, where $\Sigma_i = \text{diag}(\sigma_{i1}^2, \sigma_{i2}^2, \dots, \sigma_{ip}^2)$. Here, features are assumed to be uncorrelated within each component, but the variance is allowed to differ across dimensions. This increases the number of parameters to p per component, providing greater flexibility than the spherical model while maintaining computational efficiency. The diagonal model is especially suitable when features are independent or only weakly correlated.

Full (Unconstrained) Covariance Matrices The most flexible case is the full (unconstrained) covariance model, in which each component is assigned its own unrestricted covariance matrix Σ_i. This allows clusters to be arbitrarily rotated, stretched, and oriented in the feature space, capturing correlations between variables. Each covariance matrix requires $p(p+1)/2$ parameters due to its symmetry, making this model the most expressive but also the most computationally demanding. While full covariance models are well-suited for complex, correlated data, they are prone to overfitting when data is limited or high-dimensional.

Tied Covariance Matrices Finally, the tied covariance model can be combined with the other constraints and imposes the constraint that all components share the same covariance matrix, $\Sigma_i = \Sigma$ for all i. In this case, clusters differ only in their mean vectors and mixture weights, while their shape and orientation remain the same. The number of free parameters is reduced to the cost of the shared $\Sigma + k$ mixing weights and the centers.

2.5.3. Parameter Estimation via Expectation-Maximization

Direct maximization of $\ell(\theta)$ (Equation 2.13) is analytically intractable due to the log-sum term. The Expectation–Maximization (EM) algorithm introduced by Dempster, Laird, and Rubin [DLR77] is a general iterative procedure for maximum likelihood estimation in statistical models with latent variables or incomplete data. It resolves the optimization problem by introducing responsibilities $z_j = [z_{j1}, \dots, z_{jk}]^T$. For GMM, the EM framework specializes to:

E-step (Expectation) Compute the responsibility z_{ji}, the posterior probability that x_j originates from component i:

$$z_{ji} = \frac{\pi_i \mathcal{N}(x_j \mid \mu_i, \Sigma_i)}{\sum_{l=1}^{k} \pi_l \mathcal{N}(x_j \mid \mu_l, \Sigma_l)}. \tag{2.14}$$

M-step (Maximization) Update parameters by maximizing the expected complete-data log-likelihood:

$$\pi_i = \frac{1}{N} \sum_{j=1}^{N} z_{ji}, \tag{2.15}$$

$$\mu_i = \frac{\sum_{j=1}^{N} z_{ji} x_j}{\sum_{j=1}^{N} z_{ji}}, \tag{2.16}$$

$$\Sigma_i = \frac{\sum_{j=1}^{N} z_{ji} (x_j - \mu_i) \otimes (x_j - \mu_i)}{\sum_{j=1}^{N} z_{ji}}. \tag{2.17}$$

EM iterates between E- and M-steps until convergence. While guaranteed to increase $\ell(\theta)$ monotonically, it converges to a local optimum. The EM algorithm does not necessarily converge in finite steps, however, a stopping threshold is often used to control the number of iterations.

2.5.4. Applications and Practical Considerations

GMMs excel in scenarios requiring probabilistic density estimation. In clustering, they generalize k-means by assigning soft memberships via z_{ji} (Equation 2.14). They underpin acoustic modeling in speech recognition systems as emission distributions for Hidden Markov Models [Rab89] and speaker verification systems [RQD00]. In computer vision, they enable background subtraction based on pixel-level intensity distributions [Ziv04], and facilitate anomaly detection by identifying low-probability regions under the estimated mixture [CBK09].

A key advantage of GMM is its flexibility in modeling complex, multimodal data distributions that cannot be captured by a single Gaussian. However, their performance strongly depends on the correct specification of k, and the choice of initialization can significantly affect convergence, and Jin et al. [Jin+16] show that the results can be arbitrarily bad with random initialization. An initialization with k-means++ [AV07] or using k-means clustering to improve the quality of the initial centers can help to mitigate this problem.

It can also suffer from numerical issues such as singular covariance matrices, likelihood overfitting, or the collapse of components onto individual data points, all of which may lead to degenerate clusters. To improve robustness against such issues, Fraley and Raftery [FR07] introduces Maximum A Posteriori (MAP) estimation and Bayesian variants of GMM introduce priors over parameters that act as regularization, thereby stabilizing the estimation process.

2.6. Introduction to Tree Data Structures

A *tree* is a hierarchical data structure consisting of nodes connected by edges. At the top is the root node, from which the structure branches into subtrees of child nodes. This recursive form makes trees well-suited for representing hierarchical or relational data. Unlike arrays or linked lists, trees often support efficient organization, search, and updates in large datasets.

Tree structures are widely used across different areas of computer science. In both database and file systems, balanced trees such as B-trees support efficient indexing and retrieval [BM70]. In machine learning, decision trees [Bre+84] and their ensembles underpin many predictive models. For similarity and nearest-neighbor search, specialized variants such as Cover Trees [BKL06], KD-Trees [Ben75], Vantage-Point Trees [Yia93], and M-Trees [CPZ97] provide scalable solutions. Trees also accelerate clustering methods by pruning distance computations: k-means can be sped up with KD-trees [Kan+02; PM99], while hierarchical clustering benefits from BIRCH [ZRL96] and more recent tree-based optimizations for single- and complete-linkage methods [Sch24].

Because of their versatility, trees form a core building block for algorithms and systems ranging from compilers and routing to clustering and information retrieval. The following sections focus on the tree structures most relevant to this thesis, in particular Cover Trees and BIRCH.

2.7. Cover Tree

Beygelzimer, Kakade, and Langford [BKL06] introduced the Cover Tree, a tree-based index structure with linear memory designed to accelerate both nearest neighbor and range search. Intuitively, the Cover Tree builds a hierarchy of nested covers, where each level groups points into larger or smaller balls, with the radius halving as the tree goes deeper. In theory, the tree has an infinite number of levels with radius b^i, where b denotes the scaling factor. For theoretical analysis, $b = 2$ was used. In practice, there exists a top level where all data points lie in a single ball and a bottom level where all distinct objects are separated. Intermediate levels may be omitted if they do not provide a change to the tree structure.

Provided that the dataset has a finite expansion rate (i.e., the amount of data grows by at most a constant factor when the radius is doubled), the Cover Tree provides interesting theoretical guarantees for nearest neighbor search. In practice, it often performs well due to low overhead and inexpensive construction costs, though for low-dimensional Euclidean data, KD-Trees [Ben75] are often faster, and for general metric data, Vantage-Point trees [Yia93] may be more effective.

2.7.1. Foundations

The ball covers of the Cover Tree are restricted by the maximum radius in each level and the tree structure. An object x contained in a node n must remain within a child of n in

the next level and cannot move to another ball even if that would be closer. Every Cover Tree satisfies three invariants for its covers L_i at each level $i \in \mathbb{Z}$:

- **Nesting:** $L_i \subset L_{i-1}$.
- **Cover:** $\forall q \in L_{i-1} \exists p \in N_i \; d(p,q) \leq 2^i$ and exactly one p is the parent of q.
- **Separation:** $\forall p, q \in L_i, d(p,q) \geq 2^i$.

The routing objects $p \in L_i$ are used for navigation when searching, and at each level, balls of radius 2^i around them cover the entire dataset. Instead of storing the L_i at infinitely many levels, only those that differ from the previous are kept. To make the tree navigable from the root, we store for every $p \in L_i$ all $q \in L_{i-1}$ for which p is the parent and their distance $d(p,q)$. Since p is also its own child (distance 0) and remains a routing object in subsequent levels, any distance computed to p can be reused across all those levels. Two routing objects at the same level have at least the distance $d(p,q) > 2^i$, while descendants lie within 2^i of the routing object.

Although the scaling factor $b = 2$ is used in theoretical analysis, smaller constants such as $b = 1.2$–1.3 are often faster in practice. This scaling controls the trade-off between fan-out and depth of the tree.

2.7.2. Cover Tree Construction

A Cover Tree [BKL06] can be constructed in several ways, such as single-point insertion, bulk construction, or lazy construction. The lazy construction algorithm has different properties that do not offer advantages for clustering, so it is omitted from this thesis.

Single Insert To insert an object p, the algorithm begins at the root node, corresponding to level $i = \infty$, with the initial candidate parent set Q_∞. At each hierarchical level i, the algorithm recursively collects all children of the nodes in the current cover set Q_i and populates the next-level candidate set Q_{i-1} accordingly.

For each subsequent level, if no node $q \in Q_{i-1}$ satisfies the condition $d(p,q) < b^{i-1}$, the algorithm returns "no parent found", indicating that p cannot be inserted at a deeper level under the current constraints. Otherwise, the traversal continues to the next lower level.

If no suitable parent is identified at level $i-1$, then p is assigned as a child of some node $q \in Q_i$ such that $d(p,q) < b^i$. This ensures that p is inserted into the structure while maintaining the hierarchical and distance-based constraints defined by the parameter b and metric $d(\cdot,\cdot)$.

Bulk Construction We begin by selecting an object p as the root node, initializing the remaining dataset as the $NEAR$ set and setting the FAR set to be empty. At each recursive step, the algorithm partitions the remaining objects relative to the current pivot point p based on their distances:

- Objects within a distance of b^i from p are classified as *NEAR* and are designated as potential children in the hierarchical structure.
- Objects located beyond this radius are placed into the *FAR* set, indicating that they may be more appropriately associated with other branches or partitions within the tree.

A helper function, denoted as $\texttt{SPLIT}(d(p,\cdot), r, S)$, is employed to partition a given set of data points S into two disjoint subsets—those that lie within distance r (the near group) and those lying between r and $b \cdot r$ (the far group)—using the distance function $d(p,\cdot)$. This division is based on predefined distance thresholds.

Subtrees are constructed recursively from the accumulated *NEAR* set. As the recursion unwinds, any objects remaining in the *FAR* set that have not been incorporated into deeper levels of the structure are returned to a global pool for potential use in other branches, thereby ensuring comprehensive coverage and efficient organization of the dataset within the resulting tree.

For more details, see the original description by Beygelzimer, Kakade, and Langford [BKL06] or the explanation for a simplified version in Section 3.2.1.

2.8. BIRCH

The BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) algorithm of Zhang, Ramakrishnan, and Livny [Zha96; ZRL96; ZRL97] is a widely known and scalable cluster analysis approach in data mining. It is a multi-step procedure for numerical data that first aggregates the data in a single pass into a tree-based data structure much smaller than the original data. Concretely, it consists of four phases: (1) building the CF-Tree from the data, (2) condensing the CF-Tree into a smaller tree by rebuilding it with a larger threshold, (3) global clustering of the leaf entries using a clustering algorithm, and (4) cluster refinement by reassigning the original data points to the clusters found in the previous phase. The key contribution of BIRCH is a flexible procedure for aggregating the data into statistical summaries, called clustering features (CF), so that an informative representation is retained even when the size is reduced substantially.

The ideas of clustering features and the CF-Tree inspired extensive follow-up work. Bradley, Fayyad, and Reina [BFR98] adopts the same concept (calling them "subcluster sufficient statistics") and discusses assignments using the Mahalanobis distance; the paper omits crucial details on how to bootstrap clustering features with non-zero standard deviation. The CF-Tree has also been used for kernel density estimation by Zhang, Ramakrishnan, and Livny [ZRL99], with a threshold set on the variance to guarantee approximation quality.

The two-step clustering by Chiu et al. [Chi+01] extends BIRCH to mixed data by adding histograms over the categorical variables. If categorical variables have many levels, this becomes memory-intensive. Because BIRCH is sequentially inserting data into the CF-Tree, the tree construction can be suspended at any time. The leaves can then be processed with a clustering algorithm. When new data arrives, the tree construction is

resumed, and we trivially obtain a stream clustering algorithm (as already considered by Ganti, Gehrke, and Ramakrishnan [GGR01]). Aggarwal et al. [Agg+03] generalizes this idea in CluStream by introducing pyramidal time frames to enable the clustering of parts of the data stream by integrating temporal information. This approach was further developed in HPStream [Agg+04], which adapts CluStream to projected and subspace clustering. Cao et al. [Cao+06] uses clustering features in DenStream for density-based stream clustering to detect clusters of arbitrary shape (in contrast to earlier methods that focus on k-means-style clustering).

Breunig, Kriegel, and Sander [BKS00] adopts clustering features to perform hierarchical density-based OPTICS clustering [Ank+99] on large data and also Data Bubbles [Bre+01] to improve the quality of hierarchical clustering for high compression rates. Kranen et al. [Kra+11] combines ideas of R-trees with BIRCH clustering features to process data streams in ClusTree. Fichtenberger et al. [Fic+13] aims at improving the theoretical foundations (and hence, performance guarantees) of BIRCH by combining it with the concept of coresets in BICO. For this, it is necessary to add reference points to the clustering features, and use a strict radius threshold rather than the dynamic threshold used by BIRCH.

2.8.1. BIRCH Clustering Features

The central concept of BIRCH is a summary data structure known as Clustering Features:

$$\mathrm{CF}^{\text{BIRCH}} = (\overrightarrow{LS}, SS, n). \tag{2.18}$$

Each clustering feature represents n data points, summarized using the linear sum vector $\overrightarrow{LS} \in \mathbb{R}^d$ (with $\overrightarrow{LS}_i = \sum_x x_i$), the sum of squares $SS \in \mathbb{R}$ (a scalar $SS = \sum_i \sum_x x_i^2$) and the count $n \in \mathbb{N}$. The center of a clustering feature can be trivially computed as $\overrightarrow{LS}/n$. Using the algebraic identity $\mathrm{Var}(X) = E[X^2] - E[X]^2$, BIRCH computes the variance of a clustering feature as $\mathrm{Var}(X) = \frac{1}{n}SS - (\frac{1}{n}\sum_i LS_i)^2$. We will discuss the numerical problems with this approach in Section 3.1.1.

A new data point x can be easily integrated into the clustering feature using

$$\mathrm{CF}^{\text{BIRCH}} + x = (\overrightarrow{LS} + x, SS + \sum_i x_i^2, n + 1).$$

Because all of these are sums, two clustering features can also easily be combined (c.f., the additivity theorem in [ZRL96]) using

$$\mathrm{CF}_A^{\text{BIRCH}} + \mathrm{CF}_B^{\text{BIRCH}} = (\overrightarrow{LS}_A + \overrightarrow{LS}_B, SS_A + SS_B, n_A + n_B).$$

A single data point x can be therefore interpreted as the clustering feature containing $(x, \sum_i x_i^2, 1)$, and an empty clustering feature is simply $(0, 0, 0)$.

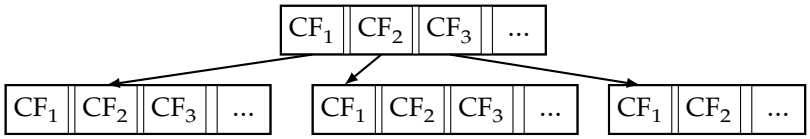


Figure 2.2.: Basic structure of a CF-Tree

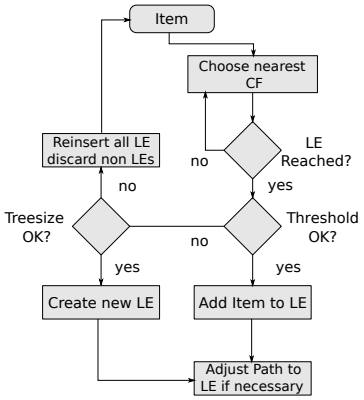


Figure 2.3.: Insertion procedure for an item, e.g., leaf entry (LE) or data object, into a CF-Tree

2.8.2. Clustering Feature Tree (CF-Tree)

Having defined how clustering features can be combined and updated, we now describe how they are stored and organized. In BIRCH, clustering features are maintained within a depth-balanced tree structure, the CF-Tree. As illustrated in Fig. 2.2, each node contains up to B clustering features, where B is the branching factor. A leaf stores a set of clustering features (each representing one or many data points), while the inner nodes store the aggregated clustering features of each of its children.

The insertion procedure is illustrated in Fig. 2.3. The tree is built by sequential insertion of data points (or, at a rebuild, the insertion of earlier clustering features). The insertion leaf is found by choosing the "nearest" clustering feature at each level (five different definitions of closeness will be discussed in Section 2.8.3). Within the leaf node, the data point is added to the best clustering feature if it is within a merging threshold; otherwise, a new clustering feature is added to the leaf. Leaves that exceed a maximum capacity are split so that an additional entry is inserted into the parent. This procedure can propagate to higher levels of the tree and cause the tree to grow when the root node overflows. If the tree exceeds the memory limit, a new tree is built with an increased merging threshold (threshold updates are discussed in Section 2.8.4) by reinserting the existing clustering features of the leaf level. After modifying a node, the aggregated clustering features along the path to the root are updated.

There are several smaller aspects to the original BIRCH algorithm such as the spilling of anomalous objects to disk and adding them to the tree in the very end. Similarly,

objects that would cause a leaf overflow can be temporarily buffered on the disk and attempted later. BIRCH also suggests an optional extra rebuild to reduce the tree size before the clustering phase. Other aspects of the original BIRCH involve low-level memory management, which is likely no longer meaningful to implement this way in modern garbage-collected languages where we do not have as much control over memory. We would like to point out that none of the implementations found in today's software appears to implement all these minor aspects of the original BIRCH, and we also do not further use or discuss these optional steps in our work.

The discussion of BIRCH in textbooks and lectures (e.g., the popular textbook of Han, Kamber, and Pei [HKP11]) often ends with the construction of the CF-Tree, although we do not yet have clusters. This is because the outstanding idea of BIRCH is that of data aggregation into clustering features, and we can run different clustering algorithms afterward. The BIRCH authors suggest a hierarchical clustering approach with their metrics D2 or D4 but notice that others could be used instead and do not provide details on additional modifications to the algorithms necessary, but implementations of BIRCH often just use the cluster feature centers as a new data set. For best results, we would want to use an algorithm that not only uses the mean of the clustering feature but that also uses the weight and variance. The weight can be fairly easily used in many algorithms, but the variance is less obvious to integrate. We detail how to use weights and variance for hierarchical clustering in Section 4.1, for *k*-means clustering in Section 5.1, and propose an adaptation of *k*-means++ initialization in Section 5.2. In Section 6.1 we will propose how to perform Gaussian Mixture Modeling to use the variance information.

2.8.3. Distance and Absorption Measures

The BIRCH algorithm uses two different measures during tree construction. The first is a distance between two clustering features, which is used to find the closest leaf in the tree. The second is an absorption criterion, used together with a threshold to decide when to add the new data to an existing cluster feature or add it as a new clustering feature. Both measures can be defined on the original data but also in terms of the clustering feature values to compute them efficiently. The distinction between distance and absorption, done in BIRCH, is rather arbitrary: all distance criteria can be used with a trivial modification for absorption and vice versa.

Distance Measures

BIRCH proposes five different distance measures enumerated as D0 to D4. The first two correspond simply to the Euclidean distance of the centers (D0) and the Manhattan distance of the centers (D1). The third, average inter-cluster distance (D2), is based on the quadratic mean distance between points of different clusters, while the average intra-cluster distance (D3) uses the quadratic mean distance within the combined cluster. Variance-increase distance (D4) is the variance of the resulting cluster minus the variance of the separated clusters. Similar ideas can be found in hierarchical clustering:

centroid linkage (D0, D1), average linkage (D2, D3), and Ward linkage (D4).

Euclidean distance:

$$\mathrm{D0}(A,B) = \|\mu_A - \mu_B\| \quad = \sqrt{\textstyle\sum_i (\mu_{A,i} - \mu_{B,i})^2} \tag{2.19}$$

Manhattan distance:

$$\mathrm{D1}(A,B) = \|\mu_A - \mu_B\|_1 = \quad \textstyle\sum_i |\mu_{A,i} - \mu_{B,i}| \tag{2.20}$$

Inter-cluster distance:

$$\mathrm{D2}(A,B) = \sqrt{\tfrac{1}{n_A n_B} \textstyle\sum_{x\in A} \sum_{y\in B} \|x-y\|^2} \tag{2.21}$$

Intra-cluster distance:

$$\mathrm{D3}(A,B) = \sqrt{\tfrac{1}{n_{AB}(n_{AB}-1)} \textstyle\sum_{x,y\in AB} \|x-y\|^2} \tag{2.22}$$

Variance-increase distance:

$$\mathrm{D4}(A,B) = \sqrt{\textstyle\sum_{x\in AB} \|x-\mu_{AB}\|^2 - \sum_{x\in A} \|x-\mu_A\|^2 - \sum_{x\in B} \|x-\mu_B\|^2} \tag{2.23}$$

Absorption Criteria

Absorption in BIRCH is based on a distance measure and a threshold. Conceptually, the threshold can be seen as a maximum radius of a cluster feature; if adding to a cluster feature would increase the radius beyond the allowed maximum, a new cluster feature is created instead of merging. Intuitively, the radius should be defined as $\max_x \|x - \mu\|$; but this value cannot be efficiently computed from the summary statistics. Instead, the "radius" can be approximated using different criteria.

In BIRCH, these criteria were defined on a single clustering feature AB; they are computed by virtually merging two clustering features and evaluating the criteria on the result. We can easily remove the distinction between distance and absorption criteria, but one may nevertheless want to choose them differently (e.g., choosing the nearest leaf by Euclidean distance but thresholding on minimum variance), as they serve a different purpose.

The first criterion proposed by Zhang, Ramakrishnan, and Livny [ZRL96] is called "radius" R (Eq. 2.24), whereas the second is the "diameter" D (Eq. 2.25). Both are not the maximum distance one would expect, but averages: the quadratic mean distance to the center is R and the quadratic mean distance of any two points is D, which happens to be the same as the $D3$ (Intra-Cluster) distance. Some implementations of BIRCH simply use the Euclidean distance (i.e., $D0$) between the two cluster centers instead. This cannot be defined in the original BIRCH "absorption" architecture, where the criteria are defined on the merged clustering feature AB, not the two features A, B before merging, but it makes the code even simpler if we postpone the computation of merging A, B. All measures here are symmetric and are defined on pairs of cluster features (not data points), as they are also used when rebuilding an existing tree.

Radius absorption criterion:

$$\mathrm{R}(AB) = \qquad \mathrm{R}(A,B) = \sqrt{\frac{1}{n_{AB}} \sum_{x \in AB} \|x - \mu_{AB}\|^2} \tag{2.24}$$

Diameter absorption criterion:

$$\mathrm{D}(AB) = \qquad \mathrm{D3}(A,B) = \sqrt{\frac{1}{n_{AB}(n_{AB}-1)} \sum_{x,y \in AB} \|x - y\|^2} \tag{2.25}$$

Euclidean absorption criterion:

$$\mathrm{D0}(A,B) = \|\mu_A - \mu_B\| \tag{2.26}$$

The values of D and R are almost identical (see Equation 3.1 below): they differ only by the factor of $\frac{2n}{n-1}$, similarly to the regular radius and diameter. Because of the way they are used in BIRCH, we cannot expect them to perform very differently. Using Euclidean distance may, for certain applications, be an interesting alternative and be closer to the original intuition of a maximum radius, at least when inserting new data points (and not inserting cluster features when rebuilding the tree, as these may have a non-zero radius on their own).

2.8.4. Updating the Absorption Threshold

When the CF-Tree exceeds the maximum size, it needs to be rebuilt with a larger threshold. The rebuilding algorithm itself is the same as the insertion algorithm, except that it inserts the leaf cluster features from the old tree instead of individual points (because of this, distance and absorption criteria need to be defined on pairs of cluster features). To choose the next threshold, BIRCH uses a heuristic that uses the average distance (using the absorption criterion) to the closest sibling of each leaf entry. The BIRCH authors claimed that this should halve the number of leave entries. But it turns out that the distribution of these distances is not symmetric but can be quite skewed, and then the reduction in tree size is often much less. This causes additional tree rebuilds, and in the worst case, only a single entry is removed each time.

As an initial threshold, we can simply use zero. Then, initially each point (except for duplicates) will be a separate cluster feature until the tree has reached the maximum size, at which point a non-trivial threshold is estimated from the data. Hence, this parameter does not have to be set by the user, but if the user has prior knowledge, choosing an appropriate threshold saves tree rebuilds and hence reduces the run time.

2.9. Numerical Stability

Numerical stability is an important aspect of scientific computing. It ensures that algorithm output matches the theory. Without it, algorithm stability can be impacted, e.g., unexpected division by zero. Sometimes numerical errors are canceling themselves, but

$\overrightarrow{LS}(X) := \sum_{x \in X} x$	$= 1.0000 + 1.0001 = 2.0001001$	floating point accuracy
$SS(X) := \sum_{x \in X} x^2$	$= 1.0000 + 1.0002 = 2.0002$	floating point accuracy
$E[X]^2 = (\overrightarrow{LS}/n)^2$	$= 1.0001001$	
$E[X^2] = SS/n$	$= 1.0001000$	
$\mathrm{Var}(X) = E[X^2] - E[X]^2$	$= -0.0000001$	catastrophic cancellation
$\sigma(X) = \sqrt{\mathrm{Var}(X)}$	$=$ *not a number* (*NaN*)	

Figure 2.4.: Catastrophic cancellation and bad rounding in BIRCH clustering features (with FP32 precision).

often they amplify. Numerical errors can impact the reproducibility and undermine the trust in a program.

2.9.1. Catastrophic Cancellation when Computing Variances

The popular way to calculate variance $\mathrm{Var}(X) = E[X^2] - E[X]^2$ is prone to the numerical problem of "catastrophic cancellation" and even amplified by using the squared values. While single-precision floating-point numbers offer 7-8 decimal digits of precision, working with squared values cuts this precision roughly in half, and we get incorrect results already for the one-dimensional input values 1.0000 and 1.0001 as seen in Fig. 2.4. In this example, we not only have catastrophic cancellation, but due to the limited floating-point precision, we even obtain a negative variance and then an undefined standard deviation.

This is a rather extreme case. Usually, the result will not be negative but just have reduced numerical accuracy, as seen if we used 1.0001 and 1.0002 or 1.0004 and 1.0005 instead, where the result would have been 0.00034527 and 0 respectively. The correct answer clearly is 0.00005, so we are seeing poor numerical accuracy here.

2.9.2. Theorem of König, Huygens, Steiner

The central theorem underlying the equation $\mathrm{Var}(X) = E[X^2] - E[X]^2$ but also much of BIRCH, and the BETULA methods presented in this thesis, is a theorem attributed in literature to Samuel König, Christiaan Huygens, and/or Jakob Steiner in the 18th century. We present here a generalized version of it for arbitrary a:

$$\sum_i (x_i - \mu)^2 = \sum_i (x_i - a)^2 - n(\mu - a)^2 \tag{2.27}$$

for $a = 0$, we obtain the "textbook" equation easily:

$$\mathrm{Var}(X) = \frac{1}{n}\sum_i (x_i - \mu)^2 = \frac{1}{n}\sum_i x_i^2 - \mu^2 = E[X^2] - E[X]^2$$

furthermore, the theorem can be rearranged to

$$\sum_i (x_i - a)^2 = \sum_i (x_i - \mu)^2 + n(\mu - a)^2 \ . \tag{2.28}$$

In the first form, Eq. 2.27, catastrophic cancellation occurs if a deviates substantially from μ. The latter form in Eq. 2.28 is safe to use (although the contribution of the second term may get lost in numerical precision if $\mu \approx a$ and $\mu \not\approx 0$). The interesting property of this equation is that it allows us to "translate" the data set, and compute the variance contribution with respect to a different center a. We will use the term "textbook" to denote the use of the numerically problematic form, using the terminology of Schubert and Gertz [SG18].

2.10. Evaluation Metrics

In order to evaluate clustering quality and compare different clustering algorithms, we employ a range of complementary measures, including metrics that require ground truth labels and metrics that assess compactness or structural quality without labels.

Adjusted Rand Index (ARI) [HA85] quantifies the similarity between two partitions (e.g., the clustering result and a ground truth labeling). It corrects for chance agreement, with values ranging from 1 (perfect agreement) to 0 (random partitions), and even negative values for particularly poor matches. It is defined as:

$$\mathrm{ARI} = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left(\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right) / \binom{n}{2}}{\frac{1}{2}\left(\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right) - \left(\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right) / \binom{n}{2}}$$

and is computed from a $k \times m$ contingency table, where k is the number of predicted clusters and m is the number of ground truth classes. Each entry n_{ij} counts the number of objects assigned to both predicted cluster i and ground truth class j. The row sums $a_i = \sum_j n_{ij}$ and column sums $b_j = \sum_i n_{ij}$ give the total number of objects in each predicted cluster and ground truth class respectively, and n denotes the total number of objects.

Sum of Squared Errors (SSE) is a classic internal quality measure defined as

$$\mathrm{SSE} = \sum_{i=1}^{n} \left(x_i - \mu_{a(i)}\right)^2,$$

where $\mu_{a(i)}$ is the centroid assigned to each object $x_i \in \mathcal{X}$ from the dataset $\mathcal{X}$. Lower SSE values indicate that clusters are compact, and the metric is directly related to the k-means objective.

Root Mean Squared Deviation (RMSD) measures the average deviation of data points from their assigned cluster centroids. It is closely related to the SSE and can be defined as:

$$\mathrm{RMSD} = \sqrt{\tfrac{1}{n}\mathrm{SSE}}.$$

Smaller values indicating tighter and more homogeneous clusters. Due to the normalization based on the number of objects, this measure is less impacted by changes in dataset size.

Log-Likelihood is particularly relevant for probabilistic clustering models such as Gaussian mixture models [Bis07]. It expresses how well the model explains the observed data, with higher values indicating a better fit. To prevent overfitting, one often considers criteria such as AIC or BIC in conjunction with the log-likelihood.

Cophenetic Correlation [SR62] evaluates hierarchical clusterings by comparing original pairwise distances with cophenetic distances derived from the dendrogram. A high correlation indicates that the tree structure faithfully represents the original dissimilarities. The similarity between two dendrograms can be computed by calculating the Pearson correlation between their cophenetic distance matrices.

2.11. Experimental Setup

Beyond metrics, it is important to distinguish between evaluating *algorithms* and evaluating their *implementations* [KSZ17]. Two implementations of the same algorithm may differ in their choice of data structures, numerical precision, initialization, or heuristics, all of which can affect both quality and runtime. For this reason, we rely on the **ELKI** framework [Sch22], a Java library offering a comprehensive collection of clustering algorithms and evaluation measures. ELKI's modular design enables consistent experimental conditions and fair comparisons both between different algorithms and between our implementations and established baselines. All experiments were conducted on an Intel Xeon CPU E5-2690 v4 processor, restricted to a single thread. To avoid artifacts from input ordering, which can have a minor impact on some algorithms, dataset points are randomly shuffled before each experiment. Most initializations are also randomized. Each experiment was repeated with 10 different fixed seeds. Results are reported as the mean over all 10 runs, with 95% confidence intervals shown as a measure of variability.

2.12. Datasets

In our experiments, we evaluate algorithms on a selection of both real-world and synthetic datasets that capture diverse domains, including images, social media, and traffic safety. These datasets vary significantly in size and dimensionality, providing a robust basis for testing clustering performance under different conditions. The dataset sizes used throughout our experiments reflect deliberate design choices rather than limitations. Single-threaded execution on fixed hardware imposes practical upper bounds on feasible data volumes, reproducibility requirements significantly constrain the total runtime budget, and several baseline methods such as HAC and GMM do not scale to large datasets, requiring dataset sizes to remain moderate for fair comparison. BETULA and Cover-means are explicitly designed to scale, but evaluating them against non-scalable baselines requires working within the constraints those baselines impose.

Real-world datasets

ALOI (Amsterdam Library of Object Images:) The ELKI Multi-View Clustering Data Sets Based on the Amsterdam Library of Object Images (ALOI) [SZ10] consist of 110 250 color images of 1000 objects, captured from multiple orientations and under varying lighting conditions. Each object is treated as a separate class. For our experiments, we use the standard RGB color histograms (uniform binning) in various dimensions.

Istanbul Twitter dataset: The Istanbul dataset consists of geo-referenced tweets from the Istanbul area, each providing latitude – longitude coordinates along with a timestamp. For this thesis, we focus on the geographic coordinates extracted from these tweets. All clustering methods in this thesis are based on the Euclidean distance; therefore, we convert the data from the geographic coordinate system into the Universal Transverse Mercator coordinate system with the help of PROJ[1] library.

UK Traffic Accidents: This is a collection of large-scale dataset published by the UK Department for Transport, containing recorded traffic accidents. Each entry provides detailed information across accidents, vehicles, and casualties, comprising roughly 70 variables. The dataset is publicly available at `http://data.gov.uk/`. We use the location data of $\approx$ 8.4M entries.

MNIST: The MNIST dataset [Den12] comprises 60,000 training and 10,000 test images of scanned handwritten digits (0-9), normalized to a consistent 28×28 pixel resolution with grayscale intensity values. For our experiments, we used an autoencoder to reduce the dimensionality for clustering.

KDDCUP04Bio: The KDDCUP04Bio dataset [CJB04] was first used for the KDD Cup 2004. It contains 145 751 entries describing 153 protein sequences. Each entry consists of 74 features.

Covertypes: This dataset [Bla98] contains information on biotope data from several wilderness areas within Colorado's Roosevelt National Forest. It consists of $\approx$ 581012 entries with 54 dimensions. The entries are a mix between continuous variables and one-hot encoded categorical variables.

Synthetic Datasets

Most of the datasets in our study are synthetically generated, since, to the best of our knowledge, no suitable real-world dataset with appropriate cluster labels is available. In particular, class labels from classification tasks are often unsuitable for clustering evaluation, as a single class may consist of multiple clusters or overlap with others. One synthetic dataset was specifically constructed to demonstrate numerical instabilities and provoke such issues. In contrast, the datasets used for runtime evaluation are well-behaved: they rarely produce strong numerical instabilities, and even suboptimal trees often still yield reasonable clustering results.

Correlation: We employ a dataset designed to study the effect of cluster separation. This dataset consists of two Gaussian clusters in $\mathbb{R}^3$ with 150 000 points in total. Both clusters share the same standard deviations, $[\frac{4}{3}, 1, \frac{3}{4}]$, and are subjected to a random

[1] https://proj.org/en/stable/

rotation. The only parameter that varies across the dataset series is the distance between the cluster centers.

For small separations, the two clusters overlap substantially, making clustering non-trivial. As the distance increases, the task becomes easier until separation is so large that the clustering becomes trivial. Beyond this point, numerical stability effects can dominate and influence the results.

Gaussian: For the data generation of our basic Gaussian dataset, we followed the general idea outlined in the BIRCH paper, where clusters were positioned on a grid, along a sine curve, or at random. While grids are useful in low-dimensional spaces and sine patterns are meaningful mainly in 2D, naive random placement frequently caused excessive cluster overlap. To mitigate this, we instead employed Halton sampling, a quasi-random sequence that yields more evenly distributed points than uniform random sampling, thereby reducing unwanted overlap. The size of each cluster is sampled from a symmetric Dirichlet distribution. To see differences between different Gaussian mixture models, we modeled the clusters from an arbitrarily rotated Gaussian distribution, with variances drawn from the uniform distribution $\mathcal{U}(1, 2)$ and scaled with the number of dimensions to keep the difficulty non-trivial for higher-dimensional versions.

Uniform: Our last synthetic dataset is simply sampled from a d-dimensional uniform distribution $\mathcal{U}(0, 2)^d$ with sizes as required for the corresponding experiment.

Chapter 3
Tree Data Structures for Accelerating Clustering

This chapter presents our contributions to accelerating clustering algorithms by leveraging tree data structures and improving their numerical properties. We modify existing tree structures to expose the data required by clustering algorithms and enhance their numerical robustness.

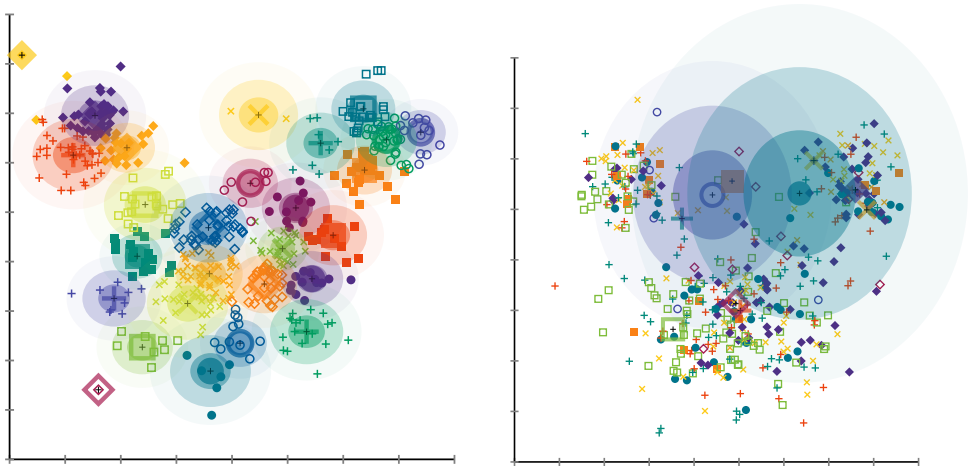
Specifically, we investigate and address numerical stability issues in the BIRCH algorithm byZhang, Ramakrishnan, and Livny [ZRL96] and extend its stored variance information to variance vectors and covariance matrices. We then discuss the modifications needed to adapt the Cover Tree introduced by Beygelzimer, Kakade, and Langford [BKL06] for use in clustering.

This chapter builds on the data structure contributions of our publications on Accelerating k-Means Clustering with Cover Trees [LS23] and BETULA [LS20; LS21; SL22], complemented by additional material and an extended evaluation.

3.1. BETULA

When studying BIRCH closely, we noticed that it is susceptible to a numerical problem known as "catastrophic cancellation", recently pointed out as a problem in clustering when using (co-)variances by Schubert and Gertz [SG18]. This arises when two large and similar floating-point values are subtracted: many bits of the significand (mantissa) cancel out, and only a few bits of valid result remain. In this section, we show how to avoid this numerical problem and demonstrate that it can arise in real data even at low dimensionality. We propose a replacement cluster feature tree (BETULA) that does not suffer from this numeric problem while retaining all functionality. Furthermore, it is often even easier to use. This structure can easily be integrated into most (if not all) derived methods.

While the BIRCH authors already mention using different clustering algorithms on the leaves, they only mention "adapted" hierarchical clustering with "D2 or D4" distance. We will discuss how to extend the variance information stored in the cluster



(a) Centered data (b) Data shifted by 10 million

Figure 3.1.: Visualization of the leaf entries of a BIRCH tree on a simple dataset with 3 Gaussian clusters.

features such that other clustering algorithms like Gaussian mixture modeling with the EM algorithm can also be used.

3.1.1. Catastrophic Cancellation in BIRCH

Fig. 3.1 shows the results of applying BIRCH to non-centered data. In Fig. 3.1a we can see how the leaf entries of BIRCH accurately represent the data with consistent variances. If the data is not centered, as shown in Fig. 3.1b, many of the leaf entries are scattered over a large area, and only two leaf entries have a valid variance. While centering may be a solution for some datasets, it is generally not possible to center each of the leaf nodes individually, and therefore the problem may still arise.

The numerical problem in BIRCH arises from the "textbook" equation for variance, $\mathrm{Var}(X) = E[X^2] - E[X]^2$. This equation, while mathematically correct, is prone to catastrophic cancellation, as discussed in Subsection 2.9.1, unless $E[X]^2 \ll E[X^2]$ holds as Schubert and Gertz [SG18] show.

In clustering, we cannot assume that all clusters are close to the origin, and the ideal leaves have a small variance and represent the data by their differences in the mean. Because of this, it may not be sufficient to center the data globally because not all cluster features will be close to 0, where the numerical accuracy is not a problem. Furthermore, we do not know the center beforehand, and in the context of BIRCH, we cannot resort to the numerically more reliable standard definition for variance, $\mathrm{Var}(X) = \frac{1}{n}\sum(x-\mu)^2$, because this requires two passes over the data set (one to find μ, then one for Var). Based on the above observation that 1.0000 and 1.0001 are already a problem, it should be obvious that we must not use BIRCH with FP32 precision. As a rule of thumb, for

FP32 input data, FP64 precision should be used, and for FP64 (double precision) input, the usage of FP128 (quad precision) would be required, which is usually not available in hardware. In any case, the increase in precision will also affect the compute performance and memory requirements negatively. Hence, we are interested in finding a more reliable variation of BIRCH.

Unfortunately, both of the original absorption criteria R and D, as well as distance measures D2–D4, are based on variance, and BIRCH relies on the "textbook" equation for computation from its cluster features:

$$\mathrm{Var}(X) = \frac{1}{2n^2}\sum_{x,y\in X}\|x-y\|^2 = \frac{1}{n}\sum_{x\in X}\|x-\mu_X\|^2\,, \tag{3.1}$$

which yields the following equalities for BIRCH:

$$n\cdot\mathrm{Var}(X) = \sum_{x\in X}\|x-\mu_X\|^2 = \frac{1}{2n}\sum_{x\in X}\|x-y\|^2 = SS - \frac{1}{n}\|\overrightarrow{LS}\|^2\,. \tag{3.2}$$

The BIRCH authors rely[1] on this property in the following equations used to compute these measures based on the data summaries. We omit D0 and D1 as they do not cause catastrophic cancellation, and we give a derivation of these equations in Appendix A.1.

Inter-cluster distance:

$$\mathrm{D2}(A,B) = \sqrt{\frac{1}{n_A n_B}(n_B SS_A + n_A SS_B \overset{\text{⚠}}{-} 2LS_A^T LS_B)} \tag{3.3}$$

Intra-cluster distance:

$$\mathrm{D3}(A,B) = \sqrt{\frac{2}{n_A+n_B-1}(SS_A + SS_B \overset{\text{⚠}}{-} \frac{1}{n_A+n_B}\|LS_A+LS_B\|^2)} \tag{3.4}$$

Variance-increase distance:

$$\mathrm{D4}(A,B) = \sqrt{\frac{1}{n_A}\|LS_A\|^2 + \frac{1}{n_B}\|LS_B\|^2 \overset{\text{⚠}}{-} \frac{1}{n_A+n_B}\|LS_A+LS_B\|^2} \tag{3.5}$$

Radius absorption criterion:

$$\mathrm{R}(AB) = \sqrt{\frac{1}{n_{AB}}(SS_{AB} \overset{\text{⚠}}{-} \frac{1}{n_{AB}}\|LS_{AB}\|^2)} \tag{3.6}$$

Diameter absorption criterion:

$$\mathrm{D}(AB) = \sqrt{\frac{2}{n_{AB}-1}(SS_{AB} \overset{\text{⚠}}{-} \frac{1}{n_{AB}}\|LS_{AB}\|^2)} \tag{3.7}$$

The subtractions flagged with a warning symbol ⚠ can suffer from catastrophic cancellation and, hence, numerical problems. The effect of the catastrophic cancellation usually leads to an underestimation of the actual variance and, hence, of the distances. Because of this, data points may be assigned to the wrong branch or node. While the result will not be completely off, it is easy to avoid these problems in the first place. It may come as unexpected that in the "variance increase" distance (D4) we only need the linear sums, but it is a consequence of the König-Huygens-Steiner theorem; see Subsection 2.9.2 that all SS terms cancel out here. More severe problems arise when using the resulting vari-

[1] The equations are not given in the main BIRCH publications of Zhang, Ramakrishnan, and Livny [ZRL96; ZRL97], but they can be found in Appendix A of the thesis of Zhang [Zha96].

ance in the subsequent steps, such as in clustering. Because many implementations of BIRCH only use the centers of the leaf entries for clustering, this has not been observed frequently. When the variance is only used during tree construction, the numerical error will often only result in a worse tree because the leaf radius is inaccurate; this problem gets more severe when we use the variance in clustering.

Much of the later work based on BIRCH is prone to the same problem in one way or another. In CF-kernel density estimation [ZRL99], the variance is bounded to guarantee approximation quality – numeric error can invalidate this guarantee. The (diagonal) Mahalanobis distance used by Bradley, Fayyad, and Reina [BFR98] divides by the standard deviation, which can become 0 or NaN due to instabilities (as seen in Fig. 2.4); the division tends to amplify the errors. CluStream [Agg+03] uses the standard deviation of the arrival times estimated with the unstable equation. HPStream [Agg+04] relies on per attribute standard deviations for subspace clustering. DenStream [Cao+06] uses the radius R to estimate density, while Data Bubbles [BKS00] rely on the standard deviation to estimate the extent. ClusTree [Kra+11] estimates the variance in this unstable way. All of these methods can easily be modified to use the improved BETULA cluster features.

3.1.2. BETULA Cluster Features

Schubert and Gertz [SG18] discuss methods for online computation of variance and covariance for weighted data, which forms the base for our approach. For this, they collect three running statistics, very similar to the three components of BIRCH cluster features:

- (i) the sum of weights,
- (ii) the weighted mean (centroid vector),
- (iii) the weighted sum of squared deviations from the mean.

Clearly (i) corresponds to n in the cluster feature, (ii) is equivalent to $\overrightarrow{LS}/n$, but (iii) is $S := \sum_x n_x \|x - \mu\|^2$ (where n_x is the weight of the data point, often simply 1). Hence, we propose the following replacement cluster feature for BETULA:

$$\mathrm{CF}^{\text{BETULA}} := (n, \mu, S) \tag{3.8}$$

where n is the aggregated weight of all data points (we allow weighted data points), μ denotes the mean vector, and S is the sum of squared deviations from the mean. A single data point of weight n_x is equivalent to a cluster feature $\mathrm{CF}_x^{\text{BETULA}} = (n_x, x, 0)$ as it has zero deviation from the mean.

Similar to the additivity theorem of BIRCH, we can efficiently combine two BETULA

$\mu(X) := \frac{1}{n}\sum_{x\in X} x$	$= 1 + \frac{1}{2}(1.0001 - 1) = 1.0000501$	incremental computation
$S(X) := \sum_{x\in X}(x - \mu(X))^2$	$= 0 + 0 + (1 - 1.0001) \cdot (1.0000501 - 1.0001)$	
	$= 0.00000000049956\!98$	floating point accuracy
$\mathrm{Var}(X) = \frac{1}{n}S(X)$	$= 0.00000000024978\!49$	floating point accuracy
$\sigma(X) = \sqrt{\mathrm{Var}(X)}$	$= 0.00004999\!78487$	

Figure 3.2.: More stable incremental computation of variance used in BETULA (with FP32 precision).

cluster features into one:

$$n_{AB} = n_A + n_B \tag{3.9}$$

$$\mu_{AB} = \mu_A + \frac{n_B}{n_{AB}}(\mu_B - \mu_A) \tag{3.10}$$

$$S_{AB} = S_A + S_B + n_B(\mu_B - \mu_A)(\mu_B - \mu_{AB}) \tag{3.11}$$

The derivation of these equations follows directly from the update equations for the weighted (co-) variance presented by Schubert and Gertz [SG18] which build upon Welford [Wel62]. Because their experiments indicate that using the sum of squared deviations, $S{:=}\sum_x n_x(x-\mu)^2$ has slight computational advantages, we follow suit. We could also have stored $\mathrm{Var}{=}S/n$ instead (we did not measure a noticeable performance difference between these two options).

Fig. 3.2 repeats the demonstration example of BIRCH with the BETULA cluster features using the incremental computation shown in Equations (3.9) to (3.11). As we can see, there is still some inevitable rounding error happening in this carefully chosen problematic example due to the numerical limits of 32-bit floating-point but compared to the input data magnitude of 1 the standard deviation error is on the order of 10^{-8}, and hence at the precision of the input data.

By using 64-bit floating-point values we can easily increase the precision; even if higher precision was required, Kahan summation would be a possible approach to further improve accuracy. However, there is little use in increasing the precision beyond the input data precision when using a method aimed at data reduction.

3.1.3. Extended Cluster Features

The last component of a cluster feature $\mathrm{CF}^{\mathrm{BETULA}}{=}(n, \mu, S)$ (the sum of squared deviations), which is a scalar in BIRCH (the sum over all components), can also be transformed into a vector of squared deviations for each axis or the matrix of cross-product deviations (to later compute the covariance matrix by division through n), which we will later use for Gaussian Mixture Modeling in Section 6.1. Similar modifications of BIRCH (that use a vector instead of a scalar) can be found in several publications (e.g., [Agg+03; Agg+04; Cao+06; HKP11; Kra+11]). In either case, we only use a scalar value (the sum of the vector or the diagonal, respectively) when constructing the CF-Tree but this ad-

ditional information can increase the quality of the clustering later. It would also be possible to define distance measures that use the individual components, but distance measures that use the individual components, e.g., the Mahalanobis distance, which is in $O(d^3)$, are expensive to compute.

Storing this additional information comes at a variable cost dependent on the form of S. For S as a vector, the memory requirement of a CF is roughly doubled but is still $O(d)$ like the scalar version. The runtime is not affected noticeably, as updating vector μ and computing distances is already in $O(d)$. When switching to the matrix of cross-product deviations, the runtime for distance calculations is still similar, but the insertion operation increases from $O(d)$ to $O(d^2)$ for updating the matrix. The memory requirement increases to $1+d+\binom{d}{2}=1+\frac{d(d+1)}{2}$ values per cluster feature.

Depending on the global clustering phase, the CF-Tree could be optimized to only use a scalar S for all inner nodes. For our experiments, we use the variation that provides the information needed in the global clustering phase and has the least overhead. When not only interested in the summarized clustering (i.e., cluster means and variances), but if we want to assign each of the input objects to a cluster partition, it becomes advisable to store object identifiers along with the cluster features. Obviously, this increases the memory by $O(n)$ for the identifiers.

3.1.4. Improved Distance Computations for BETULA Cluster Features

In BETULA cluster features, we use the mean μ instead of the linear sum because this makes the subsequent operations more efficient (and elegant). The update equations for merging CFs also involve the mean (c.f. Equation 3.11), and we can now compute the BIRCH distances in a more numerically stable way. Using BETULA cluster features $\mathrm{CF} = (n, \mu, S)$, and Eq. 3.2, we can compute the distances and absorption criteria as follows (the full derivations are given in Appendix A.2):

Euclidean distance:

$$\mathrm{D0}(A,B) = \|\mu_A - \mu_B\| \tag{3.12}$$

Manhattan distance:

$$\mathrm{D1}(A,B) = \|\mu_A - \mu_B\|_1 \tag{3.13}$$

Inter-cluster distance:

$$\mathrm{D2}(A,B) = \sqrt{\tfrac{1}{n_A}S_A + \tfrac{1}{n_B}S_B + \|\mu_A - \mu_B\|^2} \tag{3.14}$$

Intra-cluster distance:

$$\mathrm{D3}(A,B) = \sqrt{\tfrac{2}{n_{AB}(n_{AB}-1)}(n_{AB}(S_A + S_B) + n_A n_B \|\mu_A - \mu_B\|^2)} \tag{3.15}$$

Variance-increase distance:

$$\mathrm{D4}(A,B) = \sqrt{\tfrac{n_A n_B}{n_{AB}} \|\mu_A - \mu_B\|^2} \tag{3.16}$$

Radius absorption criterion:

$$\mathrm{R}_{AB} = \sqrt{\frac{1}{n_{AB}} S_{AB}} = \sqrt{\frac{1}{n_{AB}} (S_A + S_B + \frac{n_A \cdot n_B}{n_{AB}} \|\mu_A - \mu_B\|^2)} \tag{3.17}$$

Diameter absorption criterion:

$$\mathrm{D}_{AB} = \sqrt{\frac{2}{(n_{AB}-1)} S_{AB}} = \mathrm{D3}(A, B) \tag{3.18}$$

With these numerically more stable equations, we can build a CF-Tree using BETULA cluster features instead of the original BIRCH clustering features.

3.1.5. Updating the Absorption Threshold

Updating the threshold for BETULA can be handled the same as for BIRCH, see Subsection 2.8.4. Therefore, it also has the same shortcomings: it is possible to rebuild the tree and only achieve much bigger or smaller change in the number of leaf nodes than expected. Because of this, we experimented with using the median instead, which ensures that the tree is reduced in size when rebuilding. Unfortunately, this requires additional memory and experimentally was slower in our experiments, and hence we here still rely on the original BIRCH heuristic. Future work may involve using an approximate median.

3.1.6. Leaf Node Clustering

The most basic clustering strategy with CF Trees treats the cluster features (CFs) stored in leaf nodes directly as final clusters without any refinements. While this method can produce a clustering very fast, it produces inadequate results for real-world applications. It provides no direct control over the number of resulting clusters or their relative sizes. Instead, both of these properties are influenced by the randomness of the rebuilds.

Critically, due to inherent navigation errors during tree construction and continuous shifts in the CF centroids themselves as new points are inserted, the initial point-to-centroid assignments can become suboptimal without recomputing the assignments. Consequently, data points are not guaranteed to reside within the cluster whose centroid they originally approximated most closely. While this approach lacks analytical utility as a standalone clustering solution, it serves as an essential foundation for understanding subsequent clustering procedures.

3.2. Cover Tree

The Cover Tree, by Beygelzimer, Kakade, and Langford [BKL06], was originally designed for nearest neighbor search in metric spaces. While some of the concepts are the same as for clustering, some requirements are different. In nearest neighbor search, the goal is either to find the k-nearest neighbors of a point or to find all points within a certain radius of a new point. In clustering, the objective is the opposite: for each point in the dataset, find the closest cluster center, which is not in the tree. To assign as many

Algorithm 1: Construction

Input: Node $\mathcal{N} = \langle q, Q, \text{childs} \rangle$

```
1  r ← max_{p∈Q} dist(q, p)                          /* Determine maximum radius */
2  i ← ⌊log_b r⌋                                     /* Calculate correct level for radius */
3  Q_new ← {p ∈ Q | dist(q, p) < b^{i−1}}            /* Direct child node */
4  B ← Q \ Q_new  N' ← ⟨q, Q_new, ∅⟩
5  Append N' to childs(N)
6  if |Q_new| > 0 then
7  |  Construction(N')
8  while B ≠ ∅ do
9  |  p ← first(B)          /* Select any point as next routing object p */
10 |  Remove p from B
   |     /* Assign points to p that are unassigned and within radius */
11 |  P ← {p' ∈ B | dist(p, p') < b^{i−1}}
12 |  Remove P from B
13 |  N'' ← ⟨p, P, ∅⟩
14 |  Append N'' to childs(N)
15 |  if |P| > 0 then
16 |  |  Construction(N'')
```

points as early as possible, it is not only important to keep neighbors close but also the nodes on every level as tight as possible. While finding the nearest cluster might be only possible on some of the lower levels, deciding which cluster centers cannot be the closest is also already helpful information. While the original Cover Tree guarantees that direct children for a node on level i are within a radius of b^i, this bound is not guaranteed for all descendants.

Izbicki and Shelton [IS15] define a slightly simplified version of the Cover Tree in which they bring up "nearest ancestor Cover Trees" that, in essence, satisfy exactly this constraint by allowing nodes only to be below their closest ancestors on every level. They also provide "Insertion" and "Balance" algorithms that can be used to build a nearest ancestor Cover Tree object by object. With often all objects already available at the beginning of the cluster analysis, a bulk construction method is preferable.

3.2.1. Cover Tree Construction

The construction of our Cover Tree follows the original batch construction approach of Beygelzimer, Kakade, and Langford [BKL06] but with a slight adjustment to satisfy the nearest ancestor invariant of Izbicki and Shelton [IS15].

Algorithm 1 describes the batch construction of a nearest ancestor Cover Tree. The main difference between building a normal Cover Tree and a nearest ancestor one is that the far points of a node are not considered as descendants. This simplifies construction and leads to a straightforward recursive construction. First, a random object $q \in \mathcal{X}$

is picked as a routing object for the root node $N_{root} = \langle q, Q, \emptyset \rangle$ which covers all other objects $Q = \mathcal{X} \setminus q$ and has no children yet. Every Cover Tree is built with a constant scaling factor b, which is in practice usually between 1.2 and 1.3.

To split a node N, we first calculate the level i of the node by calculating the distance from q to the farthest point in Q. This makes sure that we skip unused levels. Next, all objects within b^{i-1} are assigned to node $N' = \langle q, \{p \in Q \mid \text{dist}(q,p) < b^{i-1}\}, \emptyset \rangle$ which is added to the child list of node N. After that, random points are drawn from $Q' = \{p \in Q \mid \text{dist}(q,p) \geq b^{i-1}\}$ and all remaining objects in their radius are assigned to them. Lastly, call Construction on all child nodes of N.

3.2.2. Common Modifications to the Cover Tree

The canonical formulation of the Cover Tree is not always optimal in practical applications. There exist several modifications that can be used to enhance the efficiency of Cover Tree implementations. Two significant adaptations are particularly noteworthy for optimization purposes.

In canonical Cover Tree implementations, the scaling factor b determines the radius bounds b^i for nodes on level i. However, this approach often results in very conservative radii that do not fully leverage the spatial distribution of data objects. By utilizing the exact maximum distance $r_x = \max_{q_x \in x} d(p_x, q_x)$ for node x from routing object p_x to any object $p \in x$ as the node radius (rather than the bound b^i), the node adapts dynamically to the actual data. This adjustment results in tighter distance bounds and more efficient pruning during nearest neighbor or clustering operations. The practical advantage is a reduction in unnecessary computational steps while maintaining the theoretical guarantees of the Cover Tree structure.

The second modification involves truncating the tree structure by allowing leaf nodes to contain more than one object. In the original construction algorithm [BKL06], recursion continues until every node $N \leftarrow \langle q, \emptyset, \emptyset \rangle$ contains exactly one object. However, in practice, traversing a deeper tree incurs computational overhead that may eventually outweigh the benefits of further splitting. To avoid this, the recursive splitting process can be terminated early, treating nodes that still contain multiple points as leaves. This reduces the tree depth and reduces navigation costs. Provided the leaf size is chosen appropriately, the radius r_x for such nodes remains either tightly constrained or the underlying space is sufficiently sparse, rendering individual handling of each point necessary regardless.

3.2.3. Bounds within a Cover Tree

When doing clustering, we want to avoid distance computations by assigning all objects from a node $x \subseteq \mathcal{X}$ directly to a cluster without explicitly calculating the distances to individual objects $q_x \in x$ except for the routing object p_x. For this to be possible, all objects have to be closest to the same center, c. We can bound distances using the node

radius $r_x = \max_{q_x \in x} d(p_x, q_x)$:

$$d(p_x, c_i) - r_x \leq d(q_x, c_i) \leq d(p_x, c_i) + r_x \ . \tag{3.19}$$

The triangle inequality also yields bounds on the distance of the routing object p_y to a cluster center c_i, given the distance from the parent node p_x to the cluster:

$$d(p_x, c_i) - d(p_x, p_y) \leq d(p_y, c_i) \leq d(p_x, c_i) + d(p_x, p_y) \ . \tag{3.20}$$

Combining these yields upper and lower bounds for all samples in a child node.

$$d(p_x, c_j) - d(p_x, p_y) - r_y \leq d(q_y, c_j) \leq d(p_x, c_j) + d(p_x, p_y) + r_y \ . \tag{3.21}$$

3.2.4. Adaptation for Clustering

We extend the Cover Tree with a simple bottom-up aggregation after construction to store the sum $\overrightarrow{LS}_x = \sum_{p \in x} p$ and the number of samples w_x in each node x. While this noticeably increases the memory consumption of the Cover Tree (which previously stored only object references and the distances to the parent) this allows us to update the cluster centers without accessing the individual objects. We still require only one vector per node—rather than two vectors for the bounding box as in earlier k-d tree approaches to clustering. Because of the higher fan-out, we also have fewer nodes than the k-d tree.

As described in Subsection 3.2.2 we also use a minimum node size, at which we stop building the tree and instead store all remaining points directly with a cover radius of 0. For efficiency, all such singleton nodes ($|x| = 1$) are stored more compactly, omitting storing the trivial aggregated values $\overrightarrow{LS}_x = x$ and $w_x = 1$, and the radius $r = 0$.

3.2.5. Truncated Node Clustering

In truncated node clustering, any nodes where the tree-building process stops early (see Section 3.2.2) are treated as clusters. This is a simple shortcut, but it is not a reliable way to form clusters. In particular, nodes created on the "far" side of a parent split often group data that does not really belong together, which makes the resulting clusters difficult to interpret and of limited practical value.

Furthermore, clusters display substantial size heterogeneity due to the absence of minimum size constraints, which may produce micro-clusters lacking sufficient objects to ensure statistical robustness or practical utility in subsequent analyses. Consequently, while truncated node clustering lacks real world applicability as a standalone partitioning strategy, it holds value for systematically investigating how scaling factor configurations and truncation thresholds influence runtime efficiency and the quality of leaves.

3.3. Evaluation

In this section, we look into different aspects of BETULA and the Cover Tree and examine them without the influences of other clustering algorithms.

First, we start with evaluating BETULA. We examine the different distance functions and absorption criteria that can be used with BETULA and BIRCH. Next, we look at the different combinations of branching factor and number of leaf nodes to determine viable configurations that can be used with clustering algorithms. The final analyzed BETULA parameter is the cluster feature model, responsible for defining the amount of variance information preserved in the model.

The second algorithm we investigate is the Cover Tree. We examine how different parameters influence the construction time and quality of nodes in the Cover Tree and have a more detailed look at the influence of the scaling factor on the runtime for different data characteristics.

Lastly, we compare both algorithms in terms of runtime and the quality of pre-clusters (leaves) over various data set parameters.

3.3.1. BETULA

Unless otherwise stated, BETULA experiments use the following base configuration: branching factor of 48, a maximum leaf count of 10 000, distance measure D4 (variance increase), and absorption criterion D4.

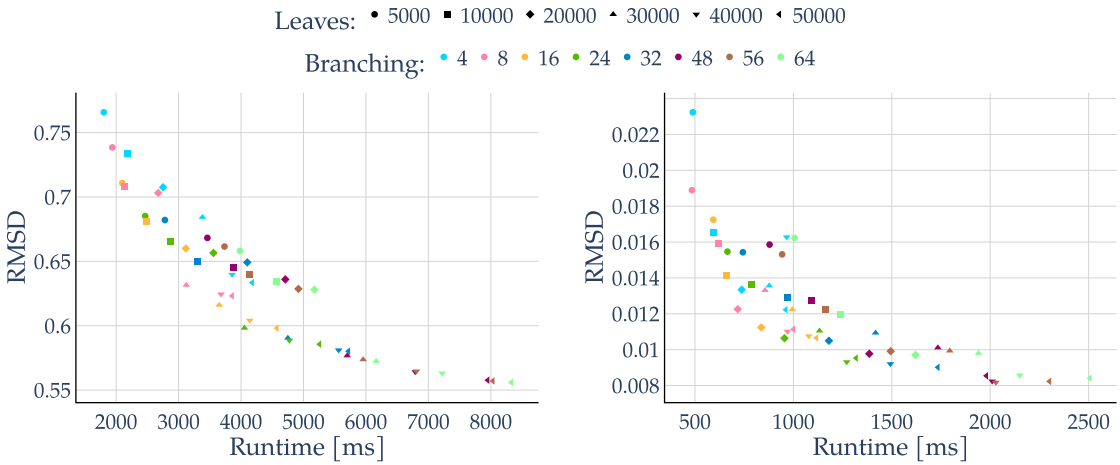
Absorption and Distance Criteria

Table 3.1 shows the residual variance and the number of leaf entries after tree construction, which are the basis for clustering with the main algorithm. For methods such as GMM and *k*-means, low variance in the cluster features is desirable, if possible, as it increases the likelihood that each leaf contributes to a single final cluster.

As expected, the variance-increase distance (D4) results in leaf entries with the lowest variance, while the Euclidean-centroid distance (D0) and inter-cluster distance (D2) also deliver solid results. However, those results are not as clear-cut as they may seem: a tree with few leaves will naturally exhibit a higher variance than a tree with more leaves. But by the concept of the CF-Tree, we do not control the exact number of leaves, but rather a maximum. Adding a single point to the CF-Tree can cause a rebuild. Such rebuilds may shrink the tree temporarily, increasing variance in preparation for further insertions. This makes it difficult to compare criteria directly, as tree size remains a major confounding factor.

The radius distance (R) and especially the Intra-cluster distance (D3) optimize for objectives other than variance minimization and therefore perform less well in this evaluation. D4, on the other hand, has the inherent advantage of being directly aligned with our evaluation measure.

When it comes to the absorption criteria, the picture is more nuanced. Using the same criterion for both distance and absorption often works well; D2 and D4 emerge as strong



(a) Gaussian (b) ALOI

Figure 3.3.: Quality and runtime tradeoff for BETULA on the Gaussian and ALOI datasets for various leaf count and branching factor combinations.

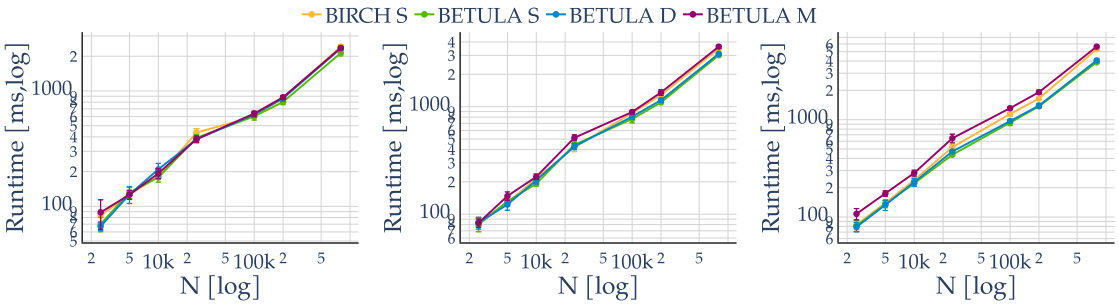
candidates. Interestingly, the original absorption criteria D3 and R are rarely optimal for minimizing variance. But because of the large effect of the tree size on the results, we cannot provide much empirical guidance. Instead, the choice of these criteria should be guided by the intended clustering objective function. In this scenario, D4 as a distance criterion and either D2 or D4 for the absorption are the most effective. Because this configuration yields relatively small trees with comparable quality, we choose D4 for both distance and absorption in the majority of the following experiments. For data on the ALOI dataset, see Appendix A.4.

Tree Size and Branching Factor

We begin by examining the parameters of the BETULA tree, as they provide ground for the algorithms that build upon it. In particular, we focus on the branching factor and the maximum number of leaf nodes. For this analysis, both the distance and absorption criteria are fixed to D4, which serves as a reliable baseline.

Ideally, we would like a fast build time and high-quality clustering. Additionally, a low leaf count benefits other applications of BETULA, as that is what directly reduces runtime. In general, larger branching factors and leaf counts improve clustering quality but come at the cost of slower tree construction.

On the Gaussian dataset, Fig. 3.3a plots runtime against RMSD for varying branching factors and leaf counts. The results indicate that the tree is built with too small of a branching factor, increasing the number of leaves does not increase the quality. Conversely, overly large branching factors only increase runtime without improving RMSD. Branching factors of ≥ 48 scale well across the entire range of leaf counts, while smaller values scale less effectively.



(a) 2D (b) 8D (c) 16D

Figure 3.4.: Runtime of the pre-clustering phase for the cluster feature models with scalar, vector, and matrix variance on the Gaussian dataset.

A similar trend appears on the ALOI dataset, shown in Fig. 3.3b. Here, the decline in RMSD is sharper than in the synthetic case, but the overall trade-offs between branching factor, leaf count, and runtime remain consistent.

Impact of Different CF Models

Regarding, the impact of CF models, such as spherical, diagonal, and covariance, affect the time needed by BETULA and BIRCH to construct the CF-Tree. In this experiment, we vary both the CF model and the dataset size.

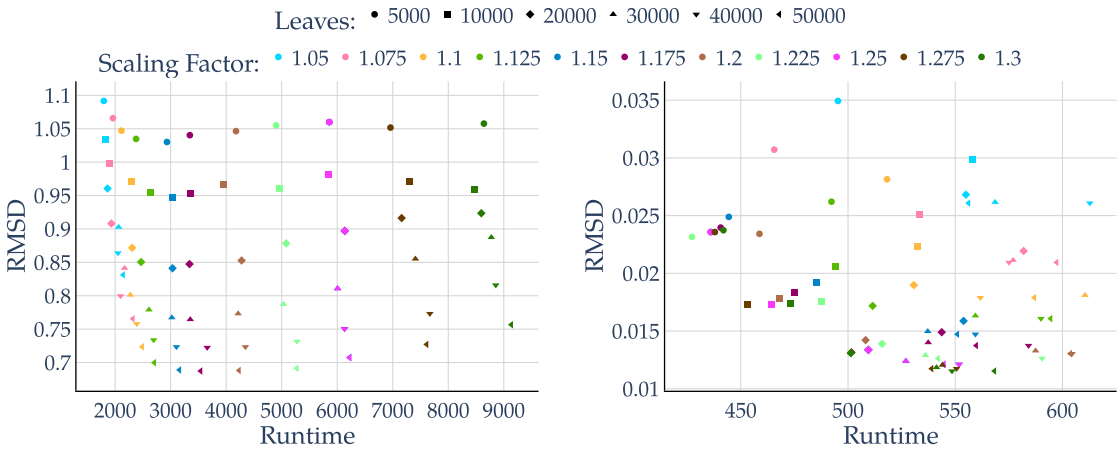
As shown in Fig. 3.4, the construction time needed for BIRCH and BETULA with spherical and diagonal CF models is not distinguishable on 2/dimensional data. For higher-dimensional data, BETULA tends to be faster than BIRCH, but both algorithms scale well with increasing dataset size. These runtime differences are not solely due to computational cost: small numerical variations can lead to structurally different trees and, consequently, to a different number of rebuilds. A slightly larger—and expected—runtime difference arises when using BETULA with the covariance CFs, which require storing and updating full covariance matrices. Updating a matrix is significantly more expensive than updating a scalar or a short vector, but these trees also retain more information about the data distribution.

Overall, the tree construction time is typically negligible compared to the subsequent clustering phase, except in cases where very small trees are built from extremely large datasets.

Takeaways

As expected, the increased information content of vector- and matrix-based CFs comes at an additional runtime cost. Therefore, we select the most efficient CF variant that still provides the necessary information for the downstream clustering algorithms.

For distance and absorption criteria, the choice is straightforward: using the same function for both generally performs well and reduces the number of free parameters.



(a) Gaussian (b) ALOI

Figure 3.5.: Quality and runtime tradeoff for the Cover Tree on the Gaussian and ALOI datasets for various scaling factor and tree size combinations.

In hierarchical clustering, we will always use the distance function most closely aligned with the chosen linkage criterion. For GMMs and *k*-means—both of which minimize variance—we adopt D4 (Variance Increase distance), since it performed among the best in our experiments (Table 3.1) and shares the same optimization goal.

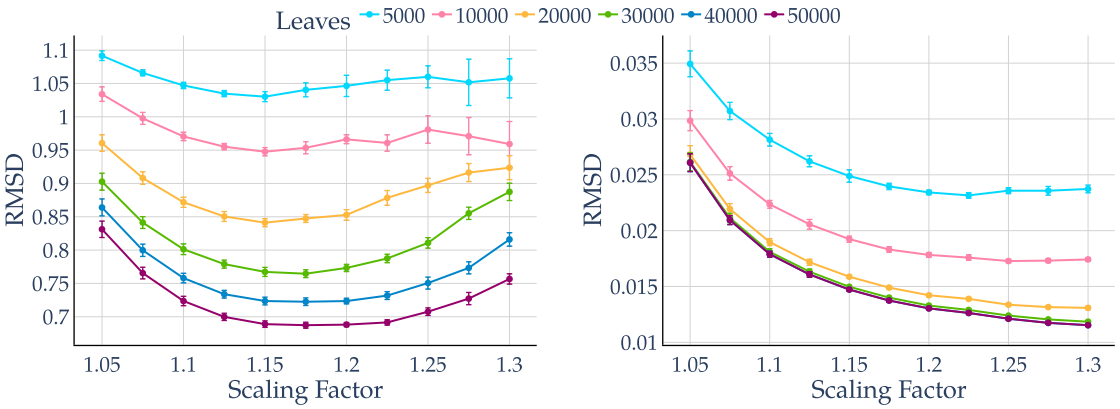
3.3.2. Cover Tree

Next, we evaluate how the CF-Tree can be enhanced by combining it with the Cover Tree, which we will later employ integrated into *k*-means clustering.

Hyperparameter Tuning

The Cover Tree behaves differently from BETULA in terms of how parameters influence quality and runtime. In BETULA, both tree width and depth affect performance, whereas in the Cover Tree, the number of leaves—closely tied to tree depth—primarily determines the quality of pre-clusters, while the scaling factor has a stronger impact on runtime.

On the Gaussian dataset Fig. 3.5a, the lowest scaling factor yields the fastest tree construction. In contrast, on the ALOI dataset in Fig. 3.5b, higher scaling factors result in faster build times. This probably comes down to the dataset-specific characteristics, particularly how the scaling factor interacts with the data distribution. For the real-world dataset, we also observe that the scaling factor influences the quality of the pre-clusters. Very low scaling factors lead to poor-quality pre-clusters.



(a) Gaussian (b) ALOI

Figure 3.6.: RMSD of Cover Tree pre-clusters for different scaling factors shown for various tree sizes on Gaussian and ALOI datasets.

Impact of the Scaling Factor on Quality

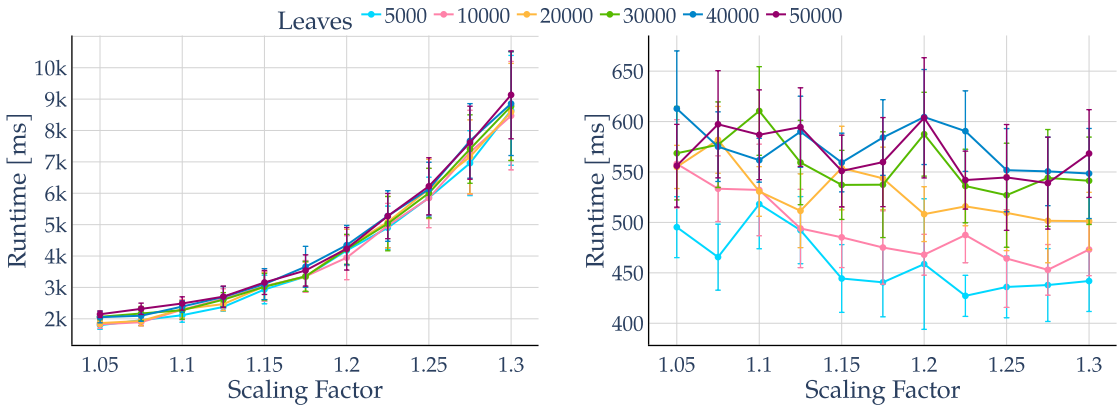
For the Cover Tree, the use of nodes as clusters is even less desirable than for BETULA. The main reason for seeking low SSE at higher levels of the tree is that such nodes are more likely to be directly assigned during the clustering phase.

As shown in Fig. 3.6a, node quality decreases with increasing branching factor when more nodes (i.e., deeper levels of the tree) are considered. An alternative perspective is that, for a given tree size or a given depth, the quality of the nodes improves with increasing scaling factor. On the ALOI dataset, Fig. 3.6b highlights that the quality is getting better with increasing scaling factor. But the difference between using 30k nodes and 50k nodes is almost negligible, which can be attributed to its size of only 105k data objects.

Impact of the Scaling Factor on Runtime

Lastly, we report the runtime over the scaling factor for various tree sizes in Fig. 3.7. On the 16-dimensional Gaussian dataset with 800 000 entries, Fig. 3.7a highlights considerable differences between the different scaling factors used to build the tree. Trees with small scaling factors are built quite fast, with almost no difference between the actual tree sizes, whereas the scaling factors commonly used in literature (typically in the $1.2 - 1.3$ range) show much worse performance. We assume this behavior is due to the more homogenous data distribution of the synthetic dataset, which leads to a higher expansion rate for the data and therefore violates the assumptions for the Cover Tree.

In Fig. 3.7b, which highlights the runtimes for the 27-dimensional ALOI dataset, we observe a vastly different picture. There is only a minimal influence of the scaling factor on the runtime performance, but for this dataset, the number of leaf nodes when stopping tree construction is an important factor for the performance, with smaller trees be-



(a) Gaussian (b) ALOI

Figure 3.7.: Runtime of Cover Tree pre-clusters for different scaling factors shown for various tree sizes on Gaussian and ALOI datasets.

ing considerably faster than bigger ones. This effect cannot be observed for the biggest trees with 30 k to 50 k leaf nodes because here the number of objects per leaf is too small.

Takeaways

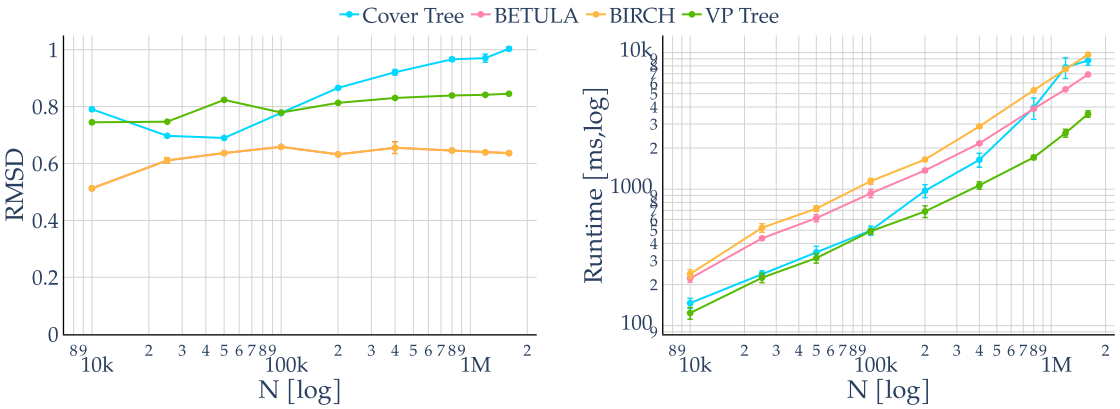
While node quality scales predictably with tree size, the influence of the scaling factor on both runtime and quality is more complex. On the ALOI dataset, choosing an appropriate scaling factor can simultaneously improve both runtime and quality. In contrast, on the synthetic Gaussian dataset, quality improves with a higher scaling factor, but this comes at a substantial runtime cost. This shows how different our two main datasets are and makes the selection of a scaling factor quite difficult.

For time-critical applications applied repeatedly to similar data, initial hyperparameter tuning for a specific dataset would likely yield the best results, but that takes additional time and resources. To ensure fairness and comparability in our experiments, we will use only a scaling factor of 1.2 in all subsequent experiments.

3.3.3. Comparison of BETULA and the Cover Tree

Lastly, we examine the partitionings produced by different trees when cut at various levels. While these partitions do not represent a meaningful clustering, they already reveal patterns that will help explain some later results when clustering algorithms are applied. Not all trees are suited in the same way for this task, so general trends are much more important than the exact results.

When not stated otherwise, all trees are built with a limit of 10 000 leaf nodes. We will use BETULA and BIRCH with the D4 (Variance-Increase) distance for both distance and absorption, and a branching factor of 48.



(a) RMSD (b) Runtime

Figure 3.8.: Runtime and quality scaling of the pre-clustering algorithms for different dataset sizes on the Gaussian dataset.

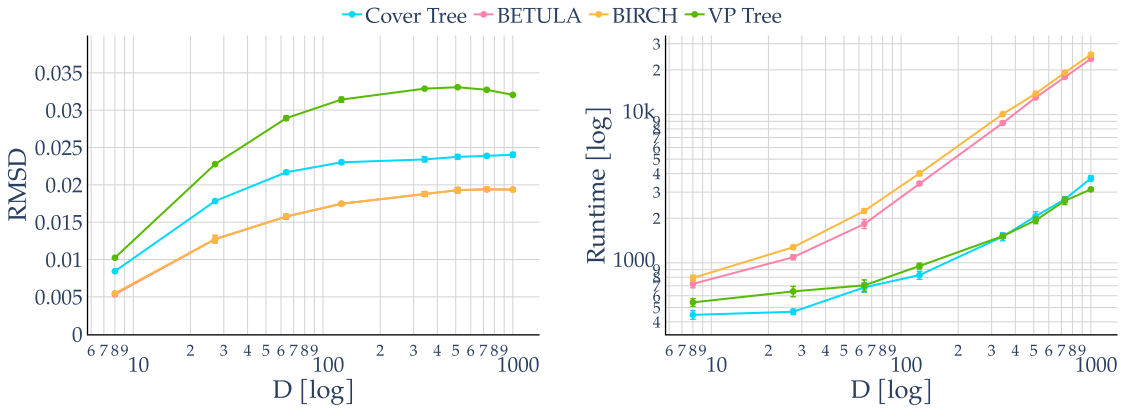
Tree Scaling with Dataset Size

First, let us investigate how the algorithms scale with the number of objects. For this comparison, we look at the Gaussian dataset with 800 k objects and 16 dimensions. In Fig. 3.8a we can observe that the average RMSD changes only slightly with the dataset size. Only the Cover Tree showcases a considerable decrease in clustering quality for larger dataset sizes. As expected, the BETULA tree exhibits the highest quality, which is identical to BIRCH because the dataset is chosen in a way that does not lead to numerical problems. It is designed for clustering, such that it minimizes the variance of the leaf nodes. In contrast, the other two trees are first and foremost intended for nearest neighbor search and offer advantages in navigation efficiency rather than clustering.

When examining the runtime in Fig. 3.8b, we can see that the VP-Tree, BIRCH, and BETULA scale very similarly, though the VP-Tree exhibits considerably less overhead. For applications like *k*-means, which scale linearly with the number of objects, even this constant overhead can be a factor for the overall performance. Regarding the Cover Tree, we observe normal behavior up to around 100 k data points, but worse scaling beyond that point. This sudden deviation in scaling behavior is related to the findings discussed in Section 3.3.2 and is therefore very dataset specific.

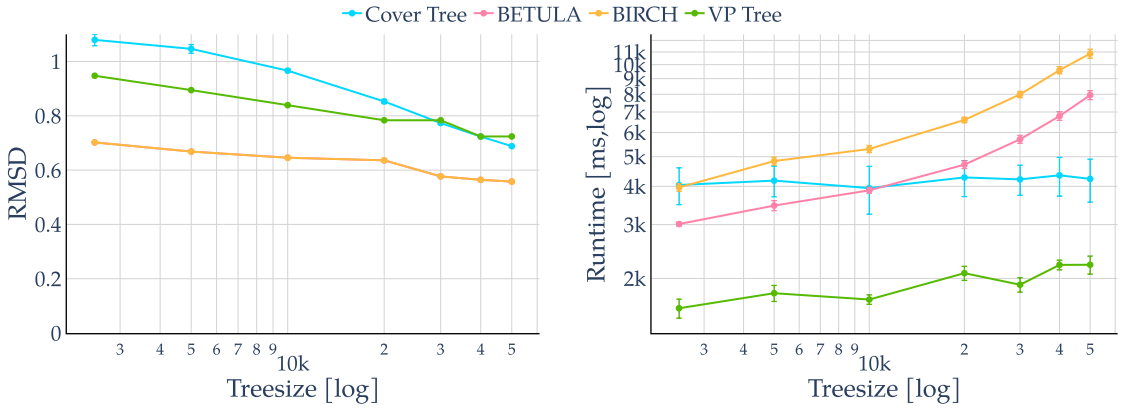
Tree Scaling with Dimensionality

Fig. 3.9a shows that on the ALOI dataset, increasing the dimensionality continues to favor BETULA and BIRCH, which is identical, in terms of quality, whereas the Cover Tree seems to work better than the VP-Tree. When examining the runtime scaling, we can observe in Fig. 3.9b that BETULA and BIRCH scale worse with increasing dimensionality compared to the Cover Tree and the VP-tree. In particular, BIRCH exhibits worse overhead as dimensionality rises than BETULA does.



(a) RMSD (b) Runtime

Figure 3.9.: Runtime and quality scaling of the pre-clustering algorithms for different dimensionalities on the ALOI dataset.



(a) RMSD (b) Runtime

Figure 3.10.: Runtime and quality scaling of the pre-clustering algorithms for different tree sizes on the Gaussian dataset.

Scaling with Tree Size

Finally, we analyze the effect of the tree size on runtime and quality when considering the leaves directly as clusters. For this experiment, we use the Gaussian dataset with 800 000 entries and 16 D. As shown in Fig. 3.10b, increasing the number of leaves leads to negligible runtime growth both for the Cover Tree and VP-Tree, with the VP-Tree outperforming the Cover Tree significantly. Most of the runtime for both of these trees is spent on splitting the large first nodes, while the overhead for additional nodes remains comparatively low. BETULA and BIRCH, however, show a noticeably stronger increase in runtime with more leaves due to their sequential insertion strategy, which induces higher navigation overhead for larger trees while reducing the number of necessary

rebuilds only slightly.

In terms of quality, Fig. 3.10a shows that the RMSD of BETULA and BIRCH is also identical for this dataset. Compared with the Cover Tree and VP-Tree, the BETULA and BIRCH produce denser leaves, which is to be expected due to their focus on minimizing the variance within the leaves and generating more uniform nodes. The Cover Tree shows the worst performance for very low leaf counts because of its node-splitting strategy. It first splits only the data points on the edges of the dataset. Only when the radius (influenced by the number of leaves) gets small enough is the area around the initial root object partitioned. At this point, the Cover Tree and VP-Tree show a comparable quality.

Table 3.1.: RMSD after pre-clustering with distance and absorption criteria combinations, and the resulting number of leaf entries on the Gaussian dataset.

	metric	RMSD / 10^{-2}		Leaves	
Dist.	Abs.	Mean	95% CI	mean	95% CI
D0	D0	78.0	±0.59	8239	±508
	D1	77.3	±0.99	7689	±619
	D2	77.3	±0.59	5677	±152
	D3	76.7	±0.75	6786	±410
	D4	72.7	±0.98	3073	±51
	R	74.0	±0.61	8981	±275
D1	D0	71.7	±1.37	9119	±729
	D1	71.4	±0.60	8907	±401
	D2	73.3	±2.24	5993	±1284
	D3	72.2	±0.75	7114	±654
	D4	68.2	±0.35	3597	±37
	R	71.5	±2.50	7966	±1344
D2	D0	80.2	±2.56	7370	±1049
	D1	78.9	±2.20	7658	±1056
	D2	71.2	±0.55	8223	±220
	D3	72.0	±0.65	8891	±207
	D4	68.1	±0.22	4390	±62
	R	71.8	±0.66	7454	±196
D3	D0	129.1	±2.42	7317	±428
	D1	129.6	±2.12	7998	±684
	D2	132.5	±3.35	8242	±446
	D3	135.1	±4.31	7646	±842
	D4	127.1	±4.74	8100	±387
	R	136.4	±4.61	8354	±418
D4	D0	66.4	±1.88	7978	±1179
	D1	64.5	±1.29	8342	±914
	D2	63.7	±0.37	9011	±182
	D3	70.1	±1.32	8734	±878
	D4	64.6	±0.25	7722	±45
	R	68.0	±0.35	8504	±191
R	D0	136.8	±3.19	7279	±540
	D1	139.2	±2.81	7870	±1064
	D2	140.6	±10.39	7166	±984
	D3	128.0	±5.19	8638	±434
	D4	127.5	±4.12	7306	±175
	R	144.0	±11.76	8116	±577

Chapter 4
Hierarchical Agglomerative Clustering

Hierarchical Agglomerative Clustering is a bottom-up clustering technique that recursively merges similar data points or clusters based on proximity metrics until a single hierarchical structure emerges. HAC remains widely adopted due to its flexibility in accommodating diverse linkage criteria (e.g., single, complete, average) and eliminating the need for pre-specifying the number of clusters, making it a cornerstone of exploratory data analysis.

However, this flexibility comes at a computational cost. HAC typically requires $O(N^2)$ memory and between $O(N^2)$ and $O(N^3)$ time, depending on the exact implementation. For large-scale data or resource-constrained environments such as embedded systems, reducing these computational demands is crucial. A common strategy to mitigate this challenge is data aggregation, where the dataset is compressed into more compact representations before clustering. One notable example is the BIRCH [ZRL96; ZRL97], which organizes the data into a compact CF-Tree before applying further clustering steps.

In this chapter, we first review how the distances in CF-Trees relate to hierarchical agglomerative clustering. Then, we present how to use BETULA to accelerate the HAC algorithm of Anderberg [And73] and the NN-Chain algorithm introduced by Murtagh [Mur83].

The content of this chapter is based on Lang and Schubert [LS20; LS21], with some additional information and an extended evaluation that now also covers an assessment of the dendrogram quality.

4.1. Hierarchical Clustering with Cluster Features

While the CF-Tree itself can be seen as a form of hierarchical clustering, its levels and internal structure are not easily interpretable. Consequently, this approach is primarily employed to aggregate data in preparation for the actual clustering process, which relies exclusively on leaf entries. Naively, one could just use the centers of the leaf entries and use a standard hierarchical clustering algorithm. Because leaves may contain a different number of points and have heterogeneous densities, treating each leaf entry as a *single*

Table 4.1.: Linkage strategy for (squared) Euclidean distances and the corresponding BIRCH distance with their objective function.

Linkage	Closed form	BIRCH Distance
UPGMA	$\frac{1}{n_A n_B} \sum_{x \in A} \sum_{y \in B} \|x - y\|^2$	$D2^2$
UPGMC	$\|\mu_A - \mu_B\|$	$D0^2$
Ward	$\frac{2 n_A n_B}{n_{AB}} \|\mu_A - \mu_B\|^2$	$2 \cdot D4^2$

point discards important statistical properties – variance and weight information – that can be valuable for accurate clustering.

An interesting observation is that the distances derived from cluster features (CF distances) closely resemble those used in common linkage criteria of hierarchical clustering (e.g., single, complete, or average linkage). We show that there is a correspondence between certain linkages and CF distances, which can be exploited for clustering by incorporating additional information stored in the cluster features besides using only the centers.

In Tab. 4.1 we summarize the identified relationships between HAC linkage strategies known from literature and BIRCH distances with their respective object function. The most obvious similarity between them can be seen when looking at the Centroid-Euclidean-Distance (D0, Eq. 2.19) and the Centroid-linkage (Eq. 2.4), which are almost the same. The differences between Ward-linkage (Eq. 2.5) and the Variance-increase-distance (D4, Eq. 2.23) are only in the notation and that D4 squared is Ward, but since BETULA internally uses squared distances for computational reasons, this difference is trivial. The last linkage that can be expressed as a BETULA distance is UPGMA, which is effectively the squared Inter-cluster-distance (D2, Eq. 2.21). This similarity becomes obvious when replacing the general equation with the one for UPGMA with the squared Euclidean distance:

$$d_{\text{UPGMA}}(A, B) = \frac{1}{n_A \cdot n_B} \sum_{a \in A} \sum_{b \in B} d(a, b) \tag{4.1}$$

$$= \frac{1}{n_A \cdot n_B} \sum_{a \in A} \sum_{b \in B} \|a - b\|^2 . \tag{4.2}$$

While WPGMA and WPGMC cannot have an exact match within the BIRCH distance functions, we may nevertheless choose D2 respectively D0 as with their "unweighted" counterparts, because of their close relationship. With this knowledge we can now meaningfully transition from cluster features into hierarchical clustering with the Lance-Williams formula by calculating the distance matrix based on the corresponding distances between the cluster features.

We can also do the opposite, and instead of using the classic linkage strategies, we can perform the following adaptation to hierarchical clustering of cluster features while using the distance functions from Section 2.8.3 instead of a separate linkage strategy. As in standard hierarchical clustering (e.g., AGNES), we find the smallest non-diagonal value in the distance matrix to find the best next merge. But instead of combining distances us-

ing the Lance-William equation, we can instead combine the corresponding two cluster features using the update equations (3.9 to 3.11) and compute new distances regarding the new CF.

For both cases (using Lance-Williams, and using CF distances), we can use the approach of Anderberg [And73] and NN-chain [Mur83] for acceleration. While the first does not improve the worst-case complexity of $O(|\text{CF}|^3)$, it typically performs closer to quadratic in runtime.

The second may yield different results for non-reducible distances (c.f. Bruynooghe [Bru77], Centroid, and Median linkage), but guarantees $O(|\text{CF}|^2)$ runtime. Furthermore, it can be implemented with only linear memory for some linkages. As the CF-Tree allows us to reduce the data to a constant size less than $O(\sqrt{N})$ respectively $O(\sqrt[3]{N})$ cluster features, we can then perform hierarchical clustering in time linear in the original data input size N and within a constant memory limit, making this useful in resource-limited data processing.

4.2. Evaluation

In this section, we compare the current HAC algorithms with their approximations based on BETULA. The algorithms considered include the HAC algorithm of Anderberg [And73] and NN-Chain [Mur83]. For Anderberg's algorithm, we also use BETULA and the unweighted initialization with BIRCH (which yields the same result as BETULA with the weighted initialization). For the NN-Chain algorithm, only BETULA is used, as BIRCH would produce the same result. For all combinations involving BETULA, we additionally evaluate variants that use CF-Tree distances, denoted with the suffix "CF".

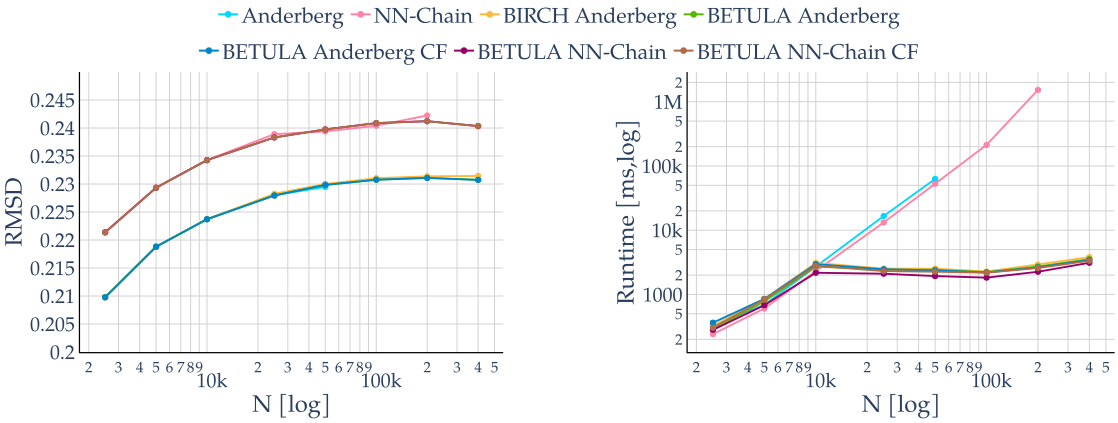
First, we examine the scaling of various HAC methods with respect to the dataset size in terms of the quality of a clustering with a fixed number of clusters and the runtime. After that, we have various views on the impact of the BETULA approximation on the dendrogram quality.

Unless stated otherwise, BETULA and BIRCH follow the base configuration introduced earlier: a branching factor of 48 and a maximum leaf count of 10 000. For the hierarchical clustering, we always use the distance measure corresponding to the chosen linkage criterion.

4.2.1. Runtime and Quality Overview

The evaluation of clustering quality reveals that BETULA and BIRCH achieve the same results as Anderberg's exact method when extracting 200 clusters (Fig. 4.1a). This pattern holds consistently across all dataset sizes where Anderberg could still be evaluated, as shown in Fig. 4.1a. For larger datasets, however, BIRCH with unweighted initialization starts to drift away slightly from BETULA.

For Ward linkage, the NN-Chain algorithm produces slightly different results with marginally reduced quality. In some cases, however, the approximation error and the



(a) RMSD (b) Runtime

Figure 4.1.: Runtime and quality scaling of the HAC algorithms with Ward linkage for different dataset sizes on the Gaussian dataset.

algorithm's own deviations cancel each other out, so that for certain dataset sizes the approximations are even negligibly better.

The runtime comparison in Fig. 4.1b shows a small overhead for tree-based methods when the dataset size is below about 10 000 objects, where each object is still represented by its own leaf. Beyond this point, the runtime of BETULA and related approximations remains almost constant, while both NN-Chain and Anderberg's algorithm continue to grow roughly quadratically with dataset size.

For BETULA, runtime is largely dominated by the hierarchical clustering phase and is therefore sensitive to the exact number of leaf nodes, which may vary due to tree rebuilds. Variants that continue to rely on CF distances achieve identical clustering results but are consistently slower than those switching to the Lance–Williams update equations. As in the non-approximated setting, NN-Chain remains slightly faster than Anderberg's algorithm.

4.2.2. Quality Based on Cophenetic Correlation

For the quality evaluation, we restrict ourselves to relatively small datasets, since computing the required pairwise distance matrices is costly. Specifically, we consider a Gaussian dataset with 8 D features and 50 000 objects, and the MNIST dataset reduced to 10 D features.

Table Table 4.2 reports the cophenetic correlation relative to the exact solution computed with Anderberg's algorithm on the Gaussian dataset with 50 000 objects in 8 dimensions, evaluated for Ward, UPGMC, and UPGMA linkage.

For Ward linkage, the unweighted BIRCH approximation deviates most strongly from the exact solution, a hardly noticeable effect when considering only the RMSD of a cross-section in Fig. 4.1a. By contrast, the other approximations as well as NN-Chain remain nearly indistinguishable from the exact solution.

Table 4.2.: Cophenetic Correlation of the HAC algorithms in relation to the exact solution provided by Anderberg's algorithm on the Gaussian dataset.

	Ward		UPGMC		UPGMA	
	MNIST	Gauss	MNIST	Gauss	MNIST	Gauss
Anderberg	1.000	1.000	1.000	1.000	1.000	1.000
BIRCH Anderberg	0.672	0.780	0.682	0.571	0.737	0.907
BETULA Anderberg	0.797	0.999	0.573	0.996	0.720	0.976
BETULA NN-Chain	0.797	0.999	0.641	0.907	0.631	0.491
NN-Chain	1.000	1.000	0.676	0.659	0.628	0.483

The UPGMC results are more striking. Once again, unweighted BIRCH performs the worst, underlining the importance of using weights during HAC initialization. BETULA's approximation of Anderberg is very close to the exact solution and, perhaps unexpectedly, even closer than NN-Chain. This suggests that dendrogram inversions occur for Euclidean linkage on the Gaussian dataset, which causes the greedy NN-Chain algorithm to diverge substantially from the exact hierarchy. BETULA combined with NN-Chain is less affected by this issue, though still inferior to the BETULA–Anderberg combination.

Finally, for UPGMA linkage, the BIRCH approximation performs well, only slightly below the quality of BETULA and the exact solution. NN-Chain, however, again produces a noticeably different clustering, making it unsuitable as a reference method for larger datasets.

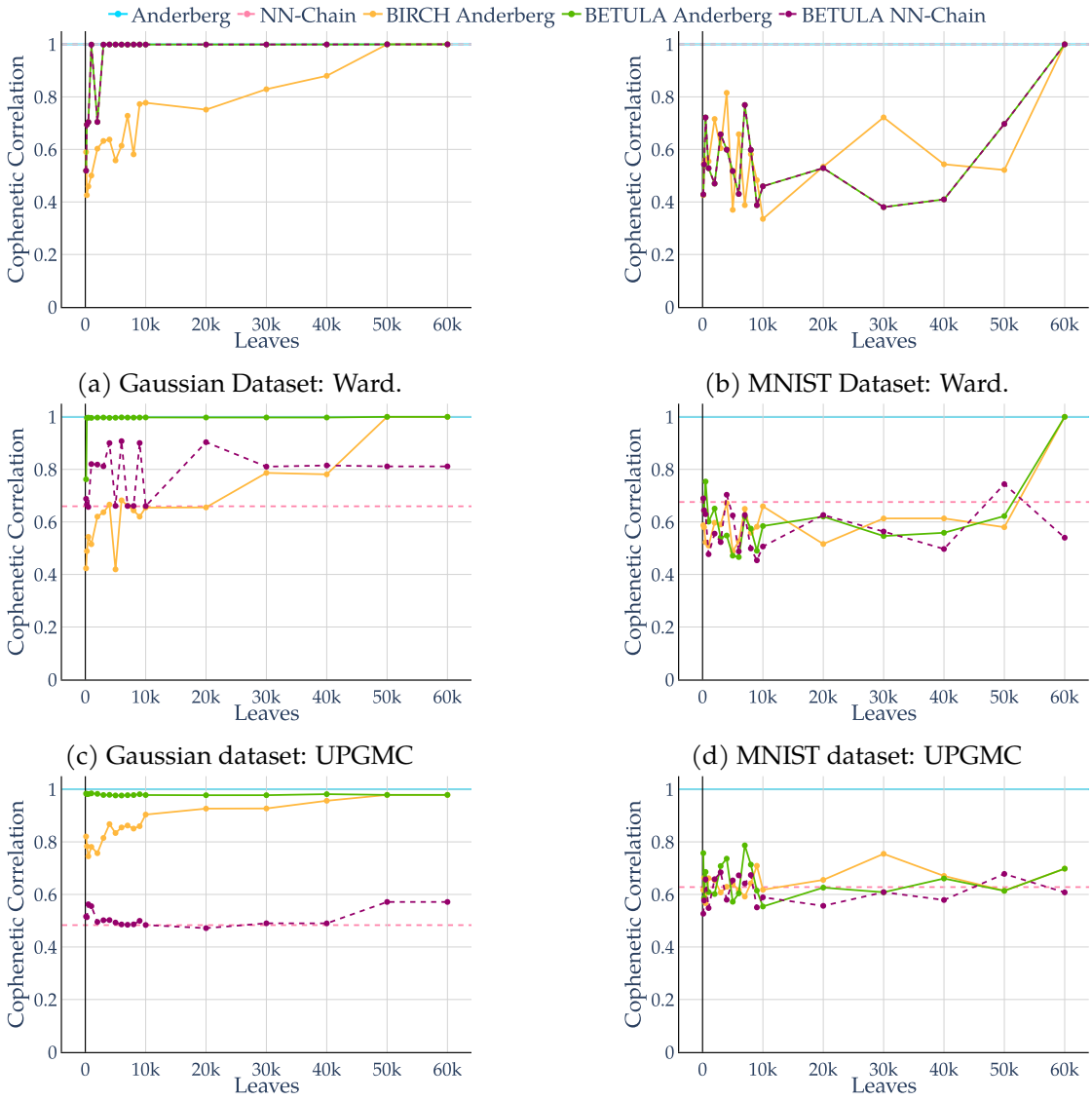
4.2.3. Leaf Count Impact on Quality

Table 4.2 reports the cophenetic correlation relative to the exact solution computed with Anderberg's algorithm on the Gaussian dataset with 50 000 objects in 8 dimensions, evaluated for Ward, UPGMC, and UPGMA linkage.

For Ward linkage, the unweighted BIRCH approximation deviates most strongly from the exact solution, an effect that was hardly visible when only considering the RMSD of a cross-section in Fig. 4.1a. By contrast, the other approximations as well as NN-Chain remain nearly indistinguishable from the exact solution.

The UPGMC results are more striking. Once again, unweighted BIRCH performs the worst, underlining the importance of using weights during HAC initialization. Here, BETULA's approximation of Anderberg stays very close to the exact solution and, perhaps unexpectedly, even surpasses NN-Chain in correlation. This suggests that dendrogram inversions occur for Euclidean linkage on the Gaussian dataset, which causes the greedy NN-Chain algorithm to diverge substantially from the exact hierarchy. BETULA combined with NN-Chain is less affected by this issue, though still inferior to using the BETULA–Anderberg combination.

Finally, for UPGMA linkage, the results differ between datasets. On the Gaussian dataset (Fig. 4.2e), the deviations from the exact solution remain small. On the MNIST



(a) Gaussian Dataset: Ward.

(b) MNIST Dataset: Ward.

(c) Gaussian dataset: UPGMC

(d) MNIST dataset: UPGMC

(e) Gaussian dataset: UPGMA

(f) MNIST dataset: UPGMA

Figure 4.2.: Cophenetic Correlation of the HAC algorithms in relation to the exact solution provided by Anderberg's algorithm for Ward, UPGMC, and UPGMA linkage on Gaussian and MNIST datasets.

dataset (Fig. 4.2f), however, both BETULA and BIRCH approximations more closely align with NN-Chain than with the exact solution. The only structural difference lies in the order of tied objects, yet this is enough to alter the cophenetic correlation.

Overall, the results demonstrate that cophenetic correlation is not immune to small structural differences in the dendrogram. Even minor variations, such as tie-breaking order, can noticeably influence the measured quality.

Chapter 5
k-Means Clustering

Among unsupervised clustering methods, k-means, often attributed to Lloyd [Llo57] is one of the most widely used. The algorithm partitions data into k non-overlapping clusters by iteratively assigning observations to the nearest cluster center (centroid), minimizing the within-cluster variance through repeated updates of cluster centers using the squared Euclidean distance. Although computationally efficient for many applications, its sensitivity to initial centroid placement often necessitates multiple restarts to avoid suboptimal solutions, thereby introducing significant computational cost in large-scale datasets where complex cluster topologies may exacerbate convergence issues.

In this chapter, we will first show how to accelerate k-means clustering with BETULA cluster features. We then introduce CF-Tree-based initialization strategies for k-means that empirically provide a comparable quality to k-means++ [AV07]. Finally, we present Cover-means, an adaptation of the Cover Tree.

The contents of this chapter were introduced in Lang and Schubert [LS21; LS23] and extended with further information. We extend the evaluation to also cover the comparison between the approximate BETULA and "exact" Cover Tree accelerations.

5.1. BETULA k-Means

The objective of k-means clustering is to minimize the sum of squared errors (SSE) when assigning each object to its nearest cluster. This is obviously closely related to the objectives of BIRCH and BETULA. Therefore, good performance can be expected when approximating the data with a CF-Tree, in particular when choosing a criterion that minimizes the variance of the leaves.

In k-means clustering, the weights of the leaves should be considered when recomputing the cluster centers. This ensures that the centers accurately reflect the distribution of the data objects within each cluster. On the other hand, variance information of the cluster features does not provide a direct benefit in standard k-means, since the algorithm does not allow partial assignment of leaves. But, this information can still be leveraged when evaluating the quality of the clustering. Let $c(\mathrm{CF})$ be the closest k-means cluster center for each cluster feature CF. We can then efficiently compute the sum of squared errors (SSE) of the k-means clustering (sometimes called "inertia") using the König-

Huygens Eq. 2.28 regarding the cluster center $c(\mathrm{CF})$ as

$$SSE = \sum_{\mathrm{CF}} \left(S_{\mathrm{CF}} + n_{\mathrm{CF}} \left\| \mu_{\mathrm{CF}} - c(\mathrm{CF}) \right\|^2 \right) \ .$$

The resulting value is an upper bound to the squared error obtained from assigning each of the original objects to the closest center. It is only an upper bound because in this computation, objects are assigned to clusters via the cluster feature they were aggregated on, and data assigned to the same cluster feature could be closest to different cluster centers. Consequently, a full reassignment of individual points to their nearest cluster centers can further reduce the error.

The actual *k*-means clustering can be done with a slightly modified version of any of the classic *k*-means algorithms, e.g. LLoyd's [Llo57], Hamerly's [Ham10] but also more recent ones by Newling and Fleuret [NF16] and Borgelt [Bor20].

For simplicity we show this exemplary for the Standard *k*-means algorithm with k clusters $\{C_1, ..., C_k\}$ and centers $c_1, ..., c_k$

1. The assignment of a cluster feature $\mathrm{CF}_x = (n_x, \mu_x, S_x)$ can be easily calculated by:

$$a(\mathrm{CF}_x) \leftarrow \underset{i \in 1,...,k}{\arg\min}\, d(\mu_x, c_i). \tag{5.1}$$

2. Each cluster center c_i can be updated using:

$$c_i \leftarrow \frac{\sum_{\mathrm{CF}_x | a(\mathrm{CF}_x)=i} n_x \mu_x}{\sum_{\mathrm{CF}_x | a(\mathrm{CF}_x)=i} n_x} \quad i \in 1, ..., k. \tag{5.2}$$

Which makes the only notable difference in the way the cluster centers are updated which is not directly affected by any of the accelerated clustering techniques.

5.2. BETULA *k*-means++ Initialization

Arthur and Vassilvitskii [AV07] introduced the popular *k*-means++ initialization technique. A very similar approach was also proposed slightly earlier by Ostrovsky et al. [Ost+06]. The key idea of both approaches is to choose objects with their probability proportional to the objects' contribution to the remaining variance, rather than uniformly sampling initial cluster centers. As a result, data objects close to previously chosen centers are unlikely to be sampled, while distant objects are more likely. This causes initial cluster centers to be less likely to be close to one another, and yields on average an $O(\log k)$ approximation to the optimum solution [AV07] even before starting the iterative optimization.

This technique can be efficiently adapted for CF-Trees by using Eq. 2.28. Given a BETULA cluster feature (n, μ, S), the canonical adaptation of *k*-means++ initialization

to CF-Trees, is to sample from the cluster features proportional to

$$S + n \cdot \min_{c \in C} \|\mu - c\|^2 ,$$

which follows directly from applying Eq. 2.28 with respect to the closest cluster center c. The procedure begins at the top of the tree, then recursively samples a path until a leaf cluster feature is reached. As in the standard k-means++, the first center can be chosen at random from the set of leaf entries, using only the weight of the CFs. For a CF-Tree with branching factor B and height $h = O(\log_B |\mathrm{CF}|)$, the selection of a cluster center now completes in $O(B \log_B |\mathrm{CF}|)$ time as opposed to $O(N)$ for regular k-means++.

However, one thing neglected by this procedure is that the chosen initial cluster centers are not individual data objects but cluster features. It is easy to see that the above computation is essentially the same as

$$n \cdot \mathrm{D2}^2(\mathrm{CF}, (1, c, 0)),$$

where the factor n arises because each cluster feature represents n objects, and the square reflects the use of squared deviations in k-means.

For a candidate cluster feature CF_A and an existing cluster feature CF_C, we hypothesize it is closer to the theoretical foundations of k-means++ initialization to use

$$n_A \cdot \mathrm{D2}^2(\mathrm{CF}_A, \mathrm{CF}_C)$$

instead, i.e., the average squared distance of each objects belonging to CF_A to any of the original objects in CF_C. Furthermore, Ostrovsky et al. [Ost+06] suggested to not picking the first element uniformly at random but with probability proportional to

$$\mathrm{SSE}(X) + N \|x - \mu(X)\|^2 .$$

This expression also follows from Eq. 2.28 with respect to the mean $\mu(X)$ of the entire data set. Notice that the cluster feature of the entire data set is simply

$$\mathrm{CF}_X = (N, \mu(X), \mathrm{SSE}(X)),$$

which can be obtained by aggregating all entries in the root node. Hence, we can choose the first entry proportional to $n_A \cdot \mathrm{D2}^2(\mathrm{CF}_A, \mathrm{CF}_X)$, making it remarkably similar to how the first and the later centers are chosen. The subsequent paths are then chosen proportional to

$$n_A \cdot \min_{\mathrm{CF}_C} \mathrm{D2}^2(\mathrm{CF}_A, \mathrm{CF}_C), \tag{5.3}$$

where CF_C are the previously chosen centers. This equation not only incorporates the distance from existing cluster centers, but also the cluster weight and the variance. Intuitively, cluster features with higher variance are more likely to be chosen than equally distant features of the same weight but lower variance, since they are more likely to

contain outlying points. We call this "intercluster" initialization.

An alternative scheme for the initialization is to choose the cluster feature CF_A for the next center based on the proportion of the increase in variance to its next nearest existing center

$$S_A + n_A \min_{c \in C} \|\mu_A - c\|^2 , \tag{5.4}$$

we call this initialization "variance" initialization.

Keep in mind that this is only an approximation to *k*-means++, as it cannot model the case where parts of the CF are closest to different cluster centers. Nonetheless, because the *k*-means++ initialization is inherently a randomized approach, and the CF-Tree is also an approximation of the data, our method is a reasonably good approximation to *k*-means++. We did not attempt to find a provable theoretical guarantees, which would likely involve finding such approximation guarantees for the CF-Tree itself.

It turned out that choosing the cluster features using the tree comes at a subtle additional cost: since we may not have traversed the relevant path yet, the closest existing cluster center must be computed for each cluster feature, unless additional caching mechanisms are introduced.

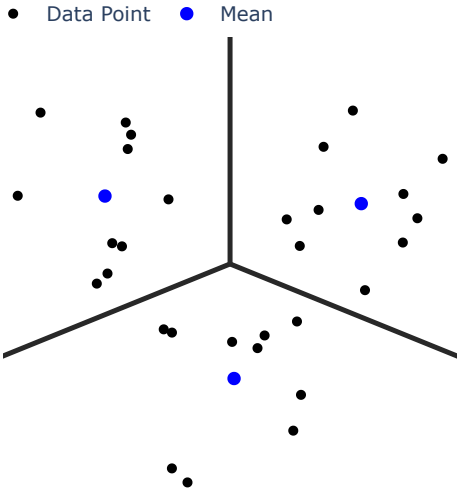
For choosing k cluster centers, we have to sample k paths of length $O(\log_B |\text{CF}|)$, each involving $O(B)$ nodes chosen proportionally using Eq. 5.4. A naive implementation yields a total cost of $O(k^2 B \log_B |\text{CF}|)$ computations of D2, a strategy denoted as "*tree*" in the experiments, while a caching strategy should allow reducing this by a factor of k to $O(kB \log_B |\text{CF}|)$.

Adding such a temporary caching to a tree structure is not very elegant (e.g., adding a map from nodes to arrays of potentially missing cached values), and becomes trivial if the candidates were a fixed array. We have not further explored optimizing the tree-based approach, as simpler alternatives experimentally performed better.

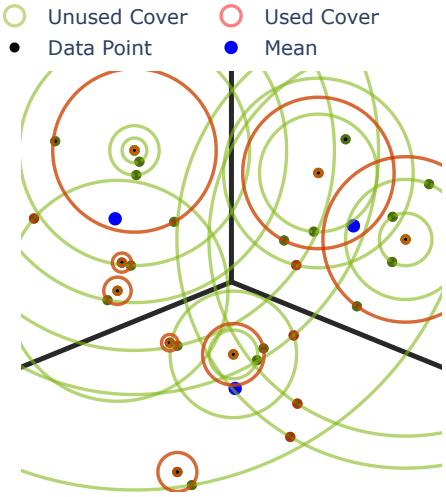
Instead of randomly sampling paths, we first expand the tree to depth $O(\log_B k)$, ensuring that at least k cluster features exist at this level, and collect all cluster features at this height in a list. This approach guarantees between k and kB candidates. For this set we can easily cache the minimum distance to all previously chosen centers, as in the regular *k*-means++ procedure, and hence choose a candidate using only $O(k^2 B)$ distance computations. We denote this strategy as "*trunk*". Alternatively, all leaf cluster features can be processed in the same way, which is likely to produce better initial objects but increases the cost to $O(k|\text{CF}|)$ distance computations. This last variant is denoted as "*leaves*" in the experiments.

5.3. Cover Tree *k*-Means

Because there exist users with the requirement for "exact" *k*-means we next propose a method that employs a variation of the Cover Tree index to accelerate *k*-means clustering by filtering candidate centers where tree nodes (representing subsets of the data) can be assigned at once, which is particularly beneficial with near-duplicate data and data



(a) Standard k-means computes the distance to every point individually, regardless of how close it is to the Voronoi partition lines.



(b) Cover Tree k-means groups points into nodes (circles). Red nodes (used covers) are traversed individually; green nodes (unused covers) are assigned as a whole.

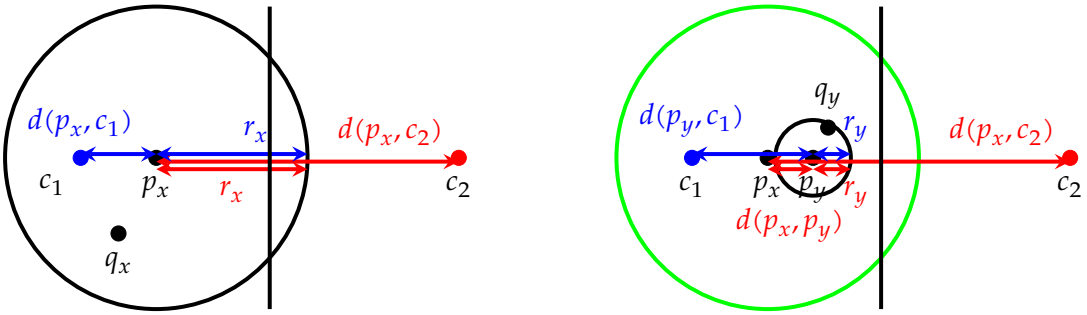
Figure 5.1.: Illustration of standard versus Cover Tree accelerated k-means with $k = 3$ clusters and their Voronoi partition.

far from the partition lines as illustrated in Fig. 5.1. Using the triangle inequality, we calculate bounds (see Section 3.2.3) to rule out cluster centers and hence reduce the number of distance computations in the assignment phase, cf. Eq. 2.6. This follows the general idea as in k-d tree–based approaches by Kanungo et al. [Kan+02] and Pelleg and Moore [PM99]. However, our approach uses fewer and smaller nodes for the tree. Additionally, as routing objects are also routing objects on all lower levels in the Cover Tree, we can reuse any distance computation to them, which further reduces the number of distance computations.

We further integrate this approach with state-of-the-art stored-bounds k-means algorithms, combining the pruning power of the Cover Tree with modern bound-based acceleration methods.

5.3.1. Calculating Distances

During initialization and when reassigning objects, we need the distance from some routing object p_x of Cover Tree node x to the centers c_i, c_j of a set of candidate clusters $C_i, C_j \in A_x$ from the set of possible nearest clusters, A_x. For this, we adapt the common lower bound in Eq. 2.10 that is used for this by other accelerated k-means algorithms, e.g., Elkan [Elk03] and Phillips [Phi02]. Unlike existing accelerated k-means algorithms, which handle only individual objects, our bounds must hold true for every object $q_x \in x$ represented by the node.



(a) Node assignment (b) Parent-child relationship

Figure 5.2.: Geometric interpretation of node assignment using the triangle inequality. The partition line separates the Voronoi regions of cluster centers c_1 and c_2.

To avoid computing all distances and to benefit from the tree structure, we rely on Eq. 3.19 and the maximum distance r_x of points in the Cover Tree node as an upper bound. This allows us to decide whether we have to calculate the distance $d(q_x, c_j)$ to new candidate clusters $C_j \in A_x$ based on the already known value of $d(p_x, c_i)$ and the inter-cluster distances $d(c_i, c_j)$:

$$\begin{aligned} d(c_i, c_j) &\geq 2d(p_x, c_i) + 2r_x \geq 2d(q_x, c_i) \\ \Rightarrow d(q_x, c_j) &\geq d(q_x, c_i) \Rightarrow C_j \notin A_x \,. \end{aligned} \tag{5.5}$$

As in prior work, the inter-cluster distances are computed and stored at the beginning of each k-means iteration, and used many times.

5.3.2. Assigning Nodes

With all relevant bounds for the routing object p_x computed, node x can be assigned to a cluster $a(x) = C_1$ if $\forall q_x \in x$, the distance to the nearest cluster center c_1 is smaller than the distance to the second-nearest cluster center c_2. This condition can be verified using the triangle inequality applied to a node, see Eq. 3.19, and is illustrated in Fig. 5.2a:

$$d(q_x, c_1) \leq d(p_x, c_1) + r_x \leq d(p_x, c_2) - r_x \leq d(q_x, c_2). \tag{5.6}$$

As shown in Fig. 5.2a, the circle of radius r_x around p_x must lie entirely within the Voronoi region of c_1, i.e., to the left of the partition line, for the node assignment to succeed. Especially for higher levels of the tree, this inequality often does not hold because nodes have a large radius r_x. Nevertheless, we may be able to eliminate candidate $C_i \in A_x$ for subsequent levels.

To rule out that a cluster C_i is the nearest to any object $q_x \in x$, the distance from its centroid c_i to the routing object $p_x \in x$ has to be larger than the upper bound on the

distance of x to its nearest cluster center. This is a generalization of Eq. 5.6:

$$d(q_x, c_1) \leq d(p_x, c_1) + r_x \leq d(p_x, c_i) - r_x \leq d(q_x, c_i). \tag{5.7}$$

As we narrow down the set of candidate cluster centers A_x, eventually only a single center remains, and the entire subtree can be assigned to the same cluster.

If we reassign the node and all its contained points to another cluster, we remove sum $\overrightarrow{LS}_y = \sum_{p \in y} p$ and the number of object w_y for all previously assigned nodes $y \subseteq x$ from their old clusters and add $\overrightarrow{LS}_x$ and w_x to the new cluster instead. By using the aggregates stored in the tree, this reassignment becomes more efficient. If we cannot assign a node, we process all child nodes recursively.

5.3.3. Recursion into Child Nodes

When moving from a parent node x to a child node $y \subset x$ in the tree, we can exploit the stored distances to the parent routing object p_x in the Cover Tree, and the radius of y. The inequality Eq. 3.21 combines the known distances to the parent routing object $d(p_x, c_i)$, the distance between parent and child routing objects $d(p_x, p_y)$, and the reduced radius of the child r_y. We can now assign the entire child to the parents' nearest cluster C_1 if $\forall q_y \in y$ the following condition holds:

$$d(q_y, c_1) \leq d(p_x, c_1) + d(p_x, p_y) + r_y \leq d(p_x, c_2) - d(p_x, p_y) - r_y \leq d(q_y, c_2). \tag{5.8}$$

Observe that for $p_x = p_y$ we obtain Eq. 5.6 with smaller radius r_y. However, the bound is only tighter when $d(p_x, p_y) \leq r_x - r_y$. Since $d(p_x, p_y) > r_y$ for $p_x \neq p_y$ and the radius r_y is typically only reduced by the chosen factor 1.2, it frequently will not be sufficient for a direct assignment. But as the tree can skip some levels, it occasionally holds and allows skipping some computations.

If the above inequality does not hold, we have to recompute at least some distances to assign y. The distance $d(p_y, c_1)$ to the nearest cluster center c_1 of the parent node is most beneficial because it can be used to tighten the inequality in Eq. 5.8 by eliminating the distance to the parent node on the left-hand side of the equation and allows us to assign y to C_1 if $\forall q_y \in y$:

$$d(q_y, c_1) \leq d(p_y, c_1) + r_y \leq d(p_x, c_2) - d(p_x, p_y) - r_y \leq d(q_y, c_2). \tag{5.9}$$

Fig. 5.2b illustrates this geometry: the larger green circle represents the parent node x with routing object p_x, and the smaller black circle the child node y with routing object p_y at distance $d(p_x, p_y)$ from the parent. The distance $d(p_x, c_2)$ already computed for the parent is reused to bound the child, avoiding recomputation. The same tightening is applied to Eq. 5.7 to prune candidate clusters before also recomputing the distances $d(q_y, c_i)$ when moving to the child node. Clusters that do not satisfy the following in-

equality can be excluded:

$$d(q_y, c_1) \leq d(p_y, c_1) + r_y \leq d(p_x, c_i) - d(p_x, p_y) - r_y \leq d(q_y, c_i). \tag{5.10}$$

After pruning cluster centers with Eq. 5.10, the same bounds and calculations are applied recursively to each of the child nodes not yet assigned a cluster.

5.4. Hybrid Cover Tree *k*-Means

When analyzing the number of distance computations performed per iteration by current state-of-the-art algorithms, such as Exponion [NF16] and Shallot [Bor20], we observe a strong correlation with the extent to which cluster centers move. As explained earlier, these algorithms drastically reduce distance computations after the first few iterations, once the centers stabilize.

In contrast, our approach is only slightly influenced by how far means move, and can already avoid distance computations in the first iteration by reducing the number of candidate cluster centers when traversing the tree. Using the stored aggregates in the tree, we can assign entire subtrees to the same cluster, but the pure tree-based approach benefits little from centers becoming stable after a few iterations.

To utilize the best of both worlds, we propose a hybrid algorithm that uses the Cover Tree only for the first few iterations and then switches to the state-of-the-art Shallot *k*-means algorithm by Borgelt [Bor20]. Any other algorithm based on Hamerly's [Ham10] bounds could be used instead, as we can efficiently obtain upper and lower bounds from our Cover Tree, which gives these algorithms an efficient start. Obtaining all $k \cdot N$ bounds for Elkan's algorithm would be more costly. Importantly, our approach is not equivalent to initializing with cluster centers obtained with the Cover Tree. Instead, we prune distance computations when computing the bounds using the tree by filtering candidate means.

For single objects, this pruning is straightforward (but we still save distance computations, as we may have a reduced set of candidates for the second-nearest center). If we assign an entire node x to a cluster C_1, we do not know the exact distance to the nearest two clusters, but we obtain upper and lower bounds by our inequalities above:

$$u_{q \in x} = d(p_x, c_1) + r_x \tag{5.11}$$

$$l_{q \in x} = d(p_x, c_2) - r_x. \tag{5.12}$$

When assigning a subtree, we can also obtain bounds for child nodes $y \subset x$ without additional distance computations:

$$u_{q \in y} = d(p_x, c_1) + d(p_x, p_y) + r_y \tag{5.13}$$

$$l_{q \in y} = d(p_x, c_2) - d(p_x, p_y) - r_y. \tag{5.14}$$

While computing $d(p_y, c_1)$ for each child node y would yield tighter bounds, we can simply leave this to the next iteration of the subsequent *k*-means algorithm. Depend-

ing on the center movements, these will need to be refined anyway, and do not need to be tight. To initialize the Shallot algorithm, we also need to provide the identity of the second-nearest cluster, i.e., the cluster for which the lower bound was obtained. While it is not guaranteed that the second-nearest cluster to the routing object in the tree is the second-nearest cluster for all points in the node, the Shallot algorithm only requires the bounds to hold. In the regular Shallot algorithm, it can also happen that the assumed second-nearest cluster changes unnoticed; this does not affect the correctness of the algorithm.

In summary, when we transition to a stored-bounds algorithm, we can reuse the bounds utilized by the Cover-means algorithm to initialize the stored bounds. These bounds are less tight, but also much less expensive to compute. Since computing initial bounds is a major bottleneck of stored-bounds approaches, this provides a significant practical advantage.

5.5. Evaluation

In this section, we evaluate the performance of the BETULA approximation of *k*-means clustering as well as the Cover Tree-based accelerated *k*-means algorithms presented in this chapter. We use the standard *k*-means algorithm of Lloyd [Llo57], as well as the common version by Hamerly [Ham10], and the more current algorithms by Newling and Fleuret [NF16] (Exponion) and Borgelt [Bor20] (Shallot). We also include the KD-Tree algorithm of Kanungo et al. [Kan+02], so we have an additional method that uses filters like Cover-means. For the Cover Tree-based algorithms, we use the name Cover-means for the plain Cover Tree variant and "Hybrid Sha" and "Hybrid Exp" for the hybrid versions with Shallot and Exponion, respectively.

First, we examine the *k*-means initialization algorithms based on BETULA, whose results are used for the rest of the evaluation. Next, we analyze runtime and distance computations over the course of the optimization process. To assess the scalability of the algorithms with respect to different dataset parameters, we employ the Gaussian and ALOI datasets. Lastly, we compare the quality and runtime of the algorithms on various real-world datasets.

Unless specified otherwise, we use the BETULA *k*-means algorithm with the leaves initialization and the BIRCH algorithm with the unweighted initialization. For both methods, we use the D4 distance criterion for both distance and absorption with 10 000 leaves and a branching factor of 48. The Cover Tree-based algorithms are constructed using a scaling factor of 1.2.

5.5.1. BETULA: *k*-Means Initialization

An important part of *k*-means and also Gaussian Mixture Modeling is the initialization of the centroids, which is usually done with *k*-means++. Because of this, we compare our *k*-means++ approximations *tree*, *trunk* and *leaves* described in Section 5.2 with the intercluster distance with *unweighted* *k*-means++ on the leaf centroids, and, for refer-

Table 5.1.: Quality of *k*-means initialization methods directly after the initialization step on the Gaussian and ALOI datasets for different dataset dimensionalities.

	Gaussian					
	2 D		8 D		32 D	
	RMSD [95% CI]/ 10^2		RMSD [95% CI]/ 10^1		RMSD [95% CI]	
Unweighted	2.214	±0.0048	3.244	±0.0063	2.126	±0.0011
Tree	2.188	±0.0066	3.102	±0.0052	2.065	±0.0007
Trunk	2.114	±0.0037	3.040	±0.0051	2.060	±0.0006
Leaves	2.111	±0.0038	3.043	±0.0047	2.060	±0.0006
Variance	2.106	±0.0035	3.009	±0.0045	2.061	±0.0006
Standard	2.098	±0.0034	3.359	±0.0045	2.378	±0.0014

	ALOI					
	8 D		27 D		64 D	
	RMSD [95% CI]/ 10^2		RMSD [95% CI]/ 10^2		RMSD [95% CI]/ 10^2	
Unweighted	1.516	±0.0149	3.131	±0.0237	3.796	±0.0212
Tree	1.247	±0.0033	2.640	±0.0041	3.248	±0.0053
Trunk	1.203	±0.0019	2.607	±0.0035	3.204	±0.0044
Leaves	1.204	±0.0020	2.607	±0.0035	3.210	±0.0042
Variance	1.199	±0.0018	2.592	±0.0035	3.193	±0.0047
Standard	1.210	±0.0017	2.659	±0.0037	3.304	±0.0042

ence, standard *k*-means++ directly on the dataset. We also evaluate leaf initialization using the *variance* criterion. The experiments are conducted on Gaussian datasets with 200 000 objects and 500 clusters, as well as on the ALOI dataset. Since the initialization step yields inherently more variable results than the final clustering, we increase the number of repetitions with different seeds to 50 for this experiment, compared to the 10 used throughout the remaining experiments, in order to obtain more reliable estimates. As a quality measure, we employ the root mean squared deviation (RMSD), which is directly related to the *k*-means objective while normalizing for the number of objects.

Table 5.1 shows that the proposed *leaves* initialization consistently outperforms the *unweighted k*-means++ applied to the leaf centroids. In contrast, the *tree* initialization does not achieve satisfactory results, which we attribute to the higher probability of random sampling along a path selecting a suboptimal subtree. The faster *trunk* initialization produces results comparable to *leaves*, unlike in Lang and Schubert [LS21], where scaling effects with dataset size were observed. The main difference is that, in the earlier experiments, the number of centers to initialize varied. This suggests that *trunk* initialization may be more effective when fewer centers are required, while *leaves* initialization is preferable for larger numbers of centers, with dimensionality playing only a minor role. Both methods also clearly outperform the commonly used *unweighted k*-means++ on the leaf entries.

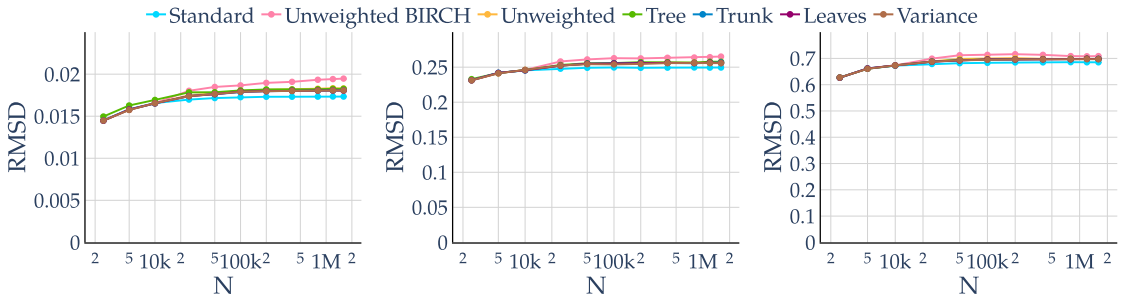
Table 5.2.: Quality of k-means initialization methods directly after the initialization step on the Gaussian and ALOI datasets for different values for k.

	Gaussian					
	$k = 100$		$k = 250$		$k = 750$	
	RMSD [95% CI / 10^3]		RMSD [95% CI / 10^3]		RMSD [95% CI / 10^3]	
Unweighted	1.082	±1.96	0.939	±1.53	0.784	±0.886
Tree	1.039	±1.51	0.886	±1.09	0.740	±0.642
Trunk	0.956	±1.27	0.867	±6.47	0.732	±0.535
Leaves	1.038	±1.46	0.882	±1.09	0.732	±0.537
Variance	1.036	±1.66	0.879	±1.12	0.731	±0.461
Standard	1.121	±1.65	0.989	±1.16	0.852	±0.628
	ALOI					
	$k = 100$		$k = 250$		$k = 750$	
	RMSD [95% CI / 10^4]		RMSD [95% CI / 10^4]		RMSD [95% CI / 10^4]	
Unweighted	4.87	±5.86	3.83	±4.52	2.75	±1.55
Tree	4.19	±1.27	3.24	±0.71	2.33	±0.36
Trunk	3.74	±1.02	3.21	±0.62	2.29	±0.28
Leaves	4.16	±0.99	3.21	±0.57	2.29	±0.27
Variance	4.16	±1.02	3.20	±0.59	2.28	±0.29
Standard	4.20	±1.09	3.25	±0.51	2.35	±0.26

What may look surprising at first is that the *leaves* initialization on the 8D and 32D datasets also demonstrates substantial improvements over standard k-means++ on the full dataset. However, this comparison is not entirely fair, since CF-Tree initializations operate on pre-aggregated data, whereas standard k-means++ must sample directly from the full dataset. The new methods therefore benefit from selecting centers from aggregated subsets, which yields better initialization performance but does not necessarily translate into superior clustering quality after optimization.

Finally, the difference between *Intercluster-* and *Variance*-based *leaves* initialization is relatively small. Nevertheless, the *Variance* criterion, being more closely aligned with the original idea of k-means++, tends to perform slightly better in certain cases.

Next, we examine the effect of varying the number of cluster centers. For this purpose, we use the Gaussian dataset with 200 000 objects in 16 D and the ALOI dataset in 27 D. Table 5.2 shows that the impact of the number of centers on the quality difference between *trunk* and *leaves* initialization is greater than that of dimensionality or even the dataset itself. For smaller numbers of centers, the *trunk* initialization appears to be better suited, while for larger numbers of centers, the *leaves* initialization performs better. Both initializations clearly outperform the standard k-means++ procedure as well as the other CF-based initialization methods.



(a) 2D (b) 8D (c) 16D

Figure 5.3.: RMSD of *k*-means initialization criteria for different dataset sizes and dimensionalities on the Gaussian dataset.

5.5.2. BETULA: *k*-Means Quality

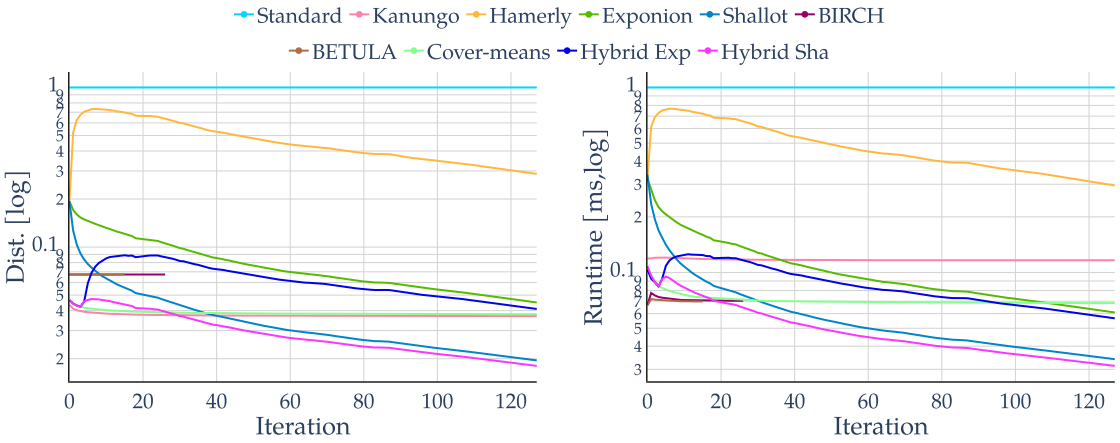
Next, we examine the scaling behavior of clustering quality after optimization. For this evaluation, we use Gaussian datasets with 2 D, 8 D, and 16 D, containing between 2500 and 1 200 000 objects. As shown in Fig. 5.3, the standard *k*-means algorithm still achieves slightly better quality after convergence. Although the differences are minimal at the beginning, the *tree* initialization consistently yields the weakest performance among the new methods. In the low-dimensional datasets with only a few points, it even performs worse than the *unweighted* initialization, suggesting that, at least for this dataset, navigating through the tree leads to too many poor choices.

As the compression ratio increases, the completely unweighted BIRCH algorithm degrades considerably compared to BETULA, even when BETULA uses the *unweighted* or *tree* initialization. Similar to the results observed directly after initialization, *trunk* and *leaves* initializations consistently provide the highest quality among aggregation-based algorithms.

5.5.3. *k*-means Convergence

In Fig. 5.4, we analyze the behavior of different *k*-means algorithms on the ALOI dataset with 64 D. For this purpose, we consider a single execution of each algorithm with a fixed seed, since otherwise the number of iterations would vary between algorithms. This serves as an example, but other iterations show the same patterns. To highlight the changes across iterations, all values are normalized by those of the standard *k*-means algorithm.

First, we examine the cumulative number of distance computations (Fig. 5.4a) over the iterations. We observe that the Hamerly algorithm already shows a substantial improvement over the standard *k*-means, particularly in the later iterations. The next improvement in terms of distance computations is *Exponion*, which is a direct extension of Hamerly's algorithm and gains most of its advantage in the first iterations. This performance can be further enhanced by the *Cover Tree Exponion Hybrid* algorithm. The



(a) Distance computations

(b) Runtime

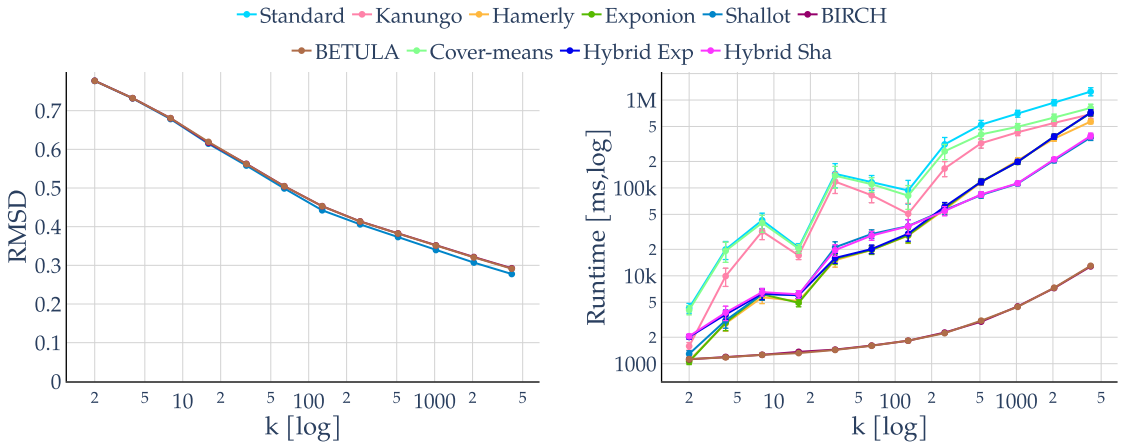
Figure 5.4.: Cumulative distance computations and runtime, in relation to the Standard *k*-means algorithms on the ALOI dataset.

Plain Cover Tree algorithm achieves fewer distance computations overall, but, similar to the standard algorithm, its performance is not strongly affected by the number of iterations. The lowest number of distance computations in our experiments is achieved by the *Shallot* and the proposed *Cover Tree Shallot Hybrid* algorithms, with the largest gains occurring in the early iterations. Finally, the approximative algorithms, being based on the standard *k*-means, follow the same overall behavior but operate on reduced data and therefore solve a less complex problem. Despite this, they require more distance computations in each iteration than the best *exact* algorithms in later iterations. Both terminate earlier due to the combination of a more central initialization (as discussed in Subsection 5.5.1) and the reduced complexity, an observation also reported by Lang and Schubert [LS21].

When we consider the cumulative runtime, again normalized by the standard algorithm (Fig. 5.4b), we observe similar algorithmic groupings as for the distance computations. The *Plain Cover Tree* algorithm has a constant runtime but incurs slightly higher overhead compared to the highly optimized *Exponion* and *Shallot* algorithms. Nonetheless, it may be particularly useful in applications where convergence is achieved early. As with distance computations, the *Cover Tree Hybrid* algorithms improve upon *Exponion* and *Shallot* by reducing the number of distance computations in the early iterations. The aggregated *k*-means algorithms with BIRCH and BETULA combine the low number of cluster features with minimal overhead compared to the standard algorithm

5.5.4. Scaling with k

For the scalability with respect to k, we consider the Uniform dataset with 8 D and 400 000 objects. In Fig. 5.5a, we examine the quality of the different algorithms. For clarity, we only include those algorithms that differ from the standard *k*-means. As



(a) RMSD (b) Runtime

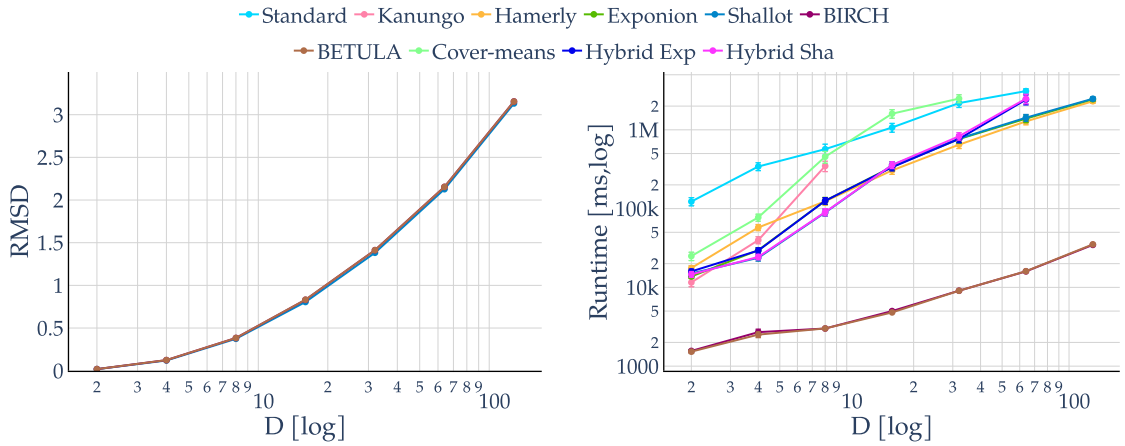
Figure 5.5.: Runtime and quality scaling for *k*-means clustering algorithms for different *k* on the Uniform dataset.

shown in the figure, the difference in quality increases with the number of cluster centers when the number of leaves is fixed. With a limit of 10 000 leaf nodes, the actual number of leaves can be much lower, resulting in fewer than two cluster features per center for $k = 5000$. At this level of compression, some loss in quality is expected, but it can be mitigated by selecting more leaves.

The more interesting scaling factor is the runtime, shown in Fig. 5.5b. Here, we observe that the *Plain Cover Tree* performs poorly with uniformly distributed data, as the absence of cluster structure reduces the effectiveness of the tree. For the exact algorithms, the *Cover Tree Hybrid* does not substantially improve the runtime of *Shallot* or *Exponion*. By contrast, the aggregated *k*-means algorithms with BIRCH and BETULA show the expected behavior: they exhibit only a relatively small increase in runtime and display lower variability compared to the "exact" algorithms. The high variability observed in the exact methods is partly due to differences in the number of iterations.

5.5.5. Scaling with Dimensionality

For the scalability with respect to dimensionality D, we again consider the Uniform dataset with 8 D and 400 000 objects. We select this dataset to avoid possible influences of changing data structure across different dimensions. In Fig. 5.6a, we observe that the difference in quality between the exact and approximative algorithms is minimal and increases only slightly with the number of dimensions. An interesting observation is the absence of any noticeable difference between the unweighted BIRCH and BETULA algorithms. Due to the uniform distribution of the data, the number of objects per cluster feature is nearly constant, meaning that weighting has little impact. This indicates that BETULA's weighting becomes most relevant for datasets with varying densities, which are more suitable for clustering.



(a) RMSD (b) Runtime

Figure 5.6.: Runtime and quality scaling for k-means clustering algorithms for dimensionalities of the Uniform dataset.

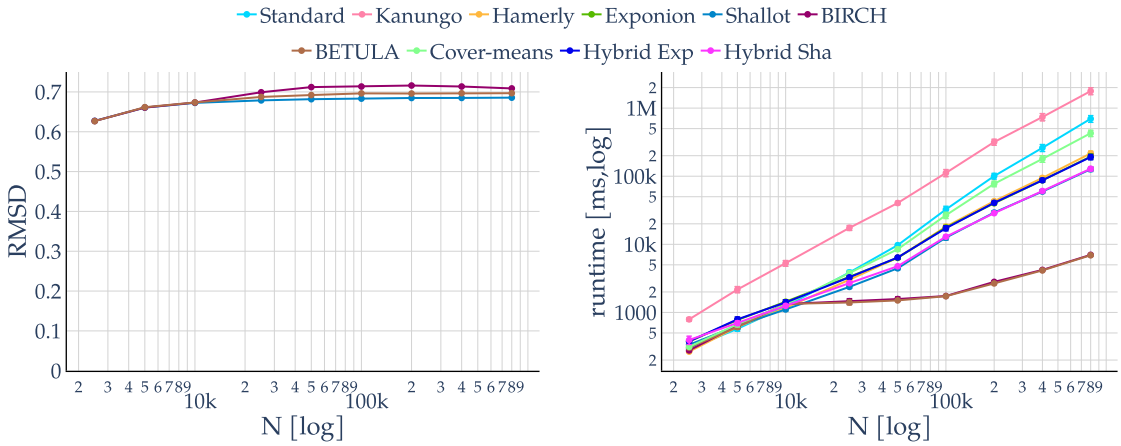
The runtime results in Fig. 5.6b are more insightful. Similar to the standard k-means algorithm, all Cover Tree algorithms time out for 128 D, with the *Plain Cover Tree* already failing at 64 D. This highlights that, in some cases, removing structural influences can negatively affect performance. Furthermore, the improved pruning strategies of the *Exponion* and *Shallot* algorithms are less effective on uniformly distributed data compared to other datasets. In contrast, the runtimes of BETULA and BIRCH increase only slightly with dimensionality, and, as expected, these algorithms remain the fastest overall.

5.5.6. Scaling with Dataset Size

Next, we analyze the runtime of the different algorithms with a varying number of objects, using the Gaussian dataset with 16 D constructed with 500 centers in mind. We select this more structured dataset because scaling with N does not alter the underlying structure, allowing us to observe how the algorithms perform on structured data.

Fig. 5.7a shows that the average RMSD increases only minimally with the number of objects. The approximative algorithms experience similarly small increases. Due to the varying density of the data, however, the unweighted BIRCH algorithm exhibits roughly twice the deviation ($\sim 4\%$) compared to the BETULA algorithm, which remains within less than $\sim 2\%$ of the standard algorithms, even for more than 1.2 M objects compacted into 10 000 leaves.

The runtime scaling with the number of objects, shown in Fig. 5.7b, largely follows expectations. The approximative algorithms exploit their strength and are the fastest overall, although their runtime still increases with N, mainly due to the initialization of the tree and the final assignment of objects to cluster centers. Depending on the objective, this final assignment step could be skipped, as discussed by Lang and Schubert [LS21].



(a) RMSD (b) Runtime

Figure 5.7.: Runtime and quality scaling for *k*-means clustering algorithms for different sizes of the Gaussian dataset.

Table 5.3.: RMSD for the *k*-means algorithms on various real-world data sets in relation to the result of the Standard *k*-means algorithm.

	Cover	KDDCUP	MNIST	ALOI	Istanbul	TrafficUK
Standard	1.000	1.000	1.000	1.000	1.000	1.000
BETULA	1.033	1.021	1.020	1.029	1.039	1.052
BIRCH	1.049	1.091	1.052	1.226	1.264	1.425

The exact algorithms remain close in terms of runtime for smaller N, but with increasing dataset size, they separate into the usual groups: the standard *k*-means and the *Plain Cover Tree* are the slowest, followed by the *Exponion*-based algorithms, with *Shallot* and the *Cover Tree Shallot Hybrid* being the fastest. It should be noted that the scaling parameter of the Cover Tree for this dataset is not ideal, as described in Section 3.3.2.

5.5.7. Quality Comparison on Further Datasets

Next, we examine the RMSD for the approximations on the real-world datasets. In Table 5.3, we can see that the RMSD for the weighted approximation with BETULA is between 3% and 4% higher than the solution with the standard *k*-means algorithm for all datasets except the TrafficUK dataset. This is the largest dataset with $\approx$ 8.4M entries, which results in 840 : 1 compression rate. The BETULA solution is 5.2% worse than the standard *k*-means result, indicating that the number of cluster features slowly becomes a limiting factor for the result quality. For the unweighted BIRCH algorithm, the RMSD is less consistent. Here we observe an increase of 5% to 10% for relatively easy Covertypes, KDDCUP, and MNIST datasets, while the RMSD for the less forgiving ALOI, Istanbul,

Table 5.4.: Runtime in seconds for the k-means algorithms on various real-world data sets. For the tree-based algorithms, we also note the runtime with a *Pre*build tree.

	Cover	KDDCUP	MNIST	ALOI	Istanbul	TrafficUK
BETULA	17.857	11.273	4.719	3.251	2.025	17.874
BIRCH	18.056	13.382	5.032	3.599	2.058	18.984
Cover-means	154.776	187.231	21.174	20.183	21.586	184.431
Exponion	130.880	128.057	15.940	16.031	16.714	197.876
Hamerly	632.856	307.023	34.033	40.486	22.308	294.042
Hybrid Exp	121.732	122.393	14.976	16.639	16.784	183.587
Hybrid Sha	90.346	77.561	8.528	13.102	15.864	181.705
Kanungo	130.280	609.795	97.523	23.501	14.261	168.625
Shallot	92.518	78.811	8.366	13.867	17.868	188.875
Standard	3618.173	898.346	86.560	134.700	91.603	1319.479
BETULA Pre.	14.441	9.695	4.074	2.648	1.174	9.473
Hybrid Sha Pre.	89.572	77.082	8.391	12.928	15.457	176.819
Kanungo Pre.	128.190	608.526	97.284	23.081	13.477	153.634

and TrafficUK datasets is between 20% and 43%.

5.5.8. Runtime Comparison on Further Datasets

When we look at the runtimes for the real-world datasets in Table 5.4, we observe that the approximate algorithms are the fastest overall, while the standard algorithm is the slowest. The approximate algorithms are between 1.7 times faster on the MNIST dataset and 9.4 times faster on the TrafficUK dataset than the fastest exact algorithm. The MNIST dataset, with 60 000 entries, is quite small, so the relatively low speedup is to be expected. The standard k-means algorithm is the slowest on all datasets, except for the MNIST dataset, where Kanungo's KD-tree-based acceleration due to dimensionality introduces too much overhead for bounding boxes and thus performs worse.

The KD-tree works very well on the 2-dimensional datasets, Istanbul and TrafficUK, where it beats all the non-approximative algorithms, but on more complex datasets, it performs worse. The Hamerly and plain Cover-means algorithms are consistently faster than the standard k-means algorithm on all datasets, but are outperformed by the algorithms that use more bounds. The expansion-based algorithms are more competitive on the low-dimensional datasets, but for most datasets, the Shallot-based algorithms perform best. Our new Cover Tree Hybrid Shallot algorithm outperforms the normal Shallot algorithm on all the datasets except for the MNIST dataset, where it lacks in runtime.

Lastly, we examine the runtime of the tree-based algorithms while omitting the time for tree construction. This comparison is relevant when performing multiple restarts on

the same dataset or comparing different k. We can see that the difference is the most pronounced for the BETULA approximations, which require time for the tree construction, as they are the only ones that use sequential insertions. Regarding the Cover Tree, the time spent on tree construction is consistently between 1%-3%. The KD-Tree showcases a small benefit for most of the datasets, except the two 2-dimensional ones, where the difference is 6% and 9%, respectively.

Chapter 6
Gaussian Mixture Modelling with BETULA

Gaussian Mixture Modeling (GMM) with the EM algorithm [DLR77] is a popular yet fairly expensive clustering method. While the algorithm is formally linear in the number of data points, it is considerably more expensive in practice due to the large number of probability density evaluations and the typically higher number of iterations required compared to k-means. This can be an issue when trying to scale GMM clustering to large datasets.

The basis for this chapter can be found in Lang and Schubert [LS20; LS21] and was extended with additional information and a new evaluation that explores the different variants in more detail.

6.1. Gaussian Mixture Modeling with BETULA Cluster Features

To scale Gaussian Mixture Model (GMM) clustering to large datasets of size N and a high number of clusters k, data aggregation can be beneficial. To achieve this, we demonstrate how extended BETULA cluster features can be employed to aggregate the data into summaries suitable for various GMM variants.

Gaussian Mixture Models can be defined under various assumptions about cluster shape and parameter sharing. In this work, we focus on three commonly used variants:

- **I-GMM (Isotropic GMM):** a spherical model with varying weights and identical variance across all dimensions,
- **D-GMM (Diagonal GMM):** a diagonal covariance model with varying weights and distinct variances along each dimension,
- **M-GMM (Multivariate GMM):** a full covariance model with varying weights, capable of representing arbitrarily oriented clusters.

If only an Isotropic Gaussian Mixture Model (I-GMM) is used, the variance information we can use is limited to scalars. In such a case, a cluster feature $\mathrm{CF}^{\text{BETULA}} = (n, \mu, S)$ with a scalar S is most appropriate. Using cluster features that store more information than the downstream clustering algorithm only negatively impacts the performance of the tree, but does not yield any quality improvements. When employing a Diagonal Gaussian Mixture Model (D-GMM), the combination of cluster features with scalar S can still be used and will result in a diagonal model, but it is advantageous to use a vector S to get the "diagonal" variance information not only through the combination of cluster features but also from within the cluster features. The same is true for a fully multivariate Gaussian Mixture Model (M-GMM), where the covariance information is a matrix. Here, the variance information from the cluster features should contain a matrix S for the covariance information.

For clustering, the main tree structure is usually discarded, and only the cluster features within the leaf nodes are kept. To optimize the runtime further, the vector or covariance matrix, respectively, could be saved just in the leaf nodes, optimizing memory usage for the rest of the tree as well as reducing some arithmetic operations for distance functions involving S. The algorithm is initialized with the adaptations of k-means++ [AV07] discussed in Section 5.2. Afterward, the model fit is performed using a simple modification of the EM algorithm.

In classic GMM, we usually process a single object at a time. When working with cluster features, however, each feature represents a group of multiple objects. For the maximization (M) step of the EM algorithm, the weight of the cluster features is trivially usable as additional weight, and its variance can be incorporated into the model variance using Eq. 3.11.

To calculate the responsibilities of each cluster in the expectation (E) step for each cluster feature, we discuss two methods. The first method, denoted "Gaussian", assumes an independent Gaussian distribution of the data in the CF. Here, we use the intersection of the cluster and the cluster features to calculate the responsibilities of each cluster for each cluster feature, which is the integral of the product of the two Gaussians with the well-known equality

$$\int_x \mathcal{N}(x \mid \mu_1, \sigma_1^2)\mathcal{N}(x \mid \mu_2, \sigma_2^2)\,\mathrm{d}x = \mathcal{N}(\mu_1 \mid \mu_2, \sigma_1^2 + \sigma_2^2) = \mathcal{N}(\mu_2 \mid \mu_1, \sigma_1^2 + \sigma_2^2)\ . \quad (6.1)$$

It can be argued that the above assumption that the data within a cluster feature is normally distributed is not justified, giving way to a much simpler approximation. The second method, denoted "point", assumes that the effect of taking the variance within cluster features into consideration is small enough to be neglected. With this assumption, the responsibilities of each cluster can be calculated as $\mathcal{N}(\mu_{CF} \mid \mu, \sigma^2)$ like the original GMM, using only the mean of the cluster features here. Using the law of total probability, for both methods, these responsibilities are normalized to sum to 1, exactly as in the usual EM procedure.

While both approaches are much faster than normal GMM, the "point" approach is much faster than the "Gaussian" approach for the multivariate Gaussian model. The reason for this efficiency difference is that in Eq. 6.1 we get a different Gaussian for

every combination of cluster and cluster feature. As a result, the computation of the responsibility then involves $|\text{CF}| \cdot k$ matrix inversions per iteration. The simpler "point" approach only needs k matrix inversions – one per cluster – like the normal GMM-EM algorithm. Both approaches yield a clustering of similar quality, so for most data sets the "point" approach is the preferred approach.

In the experiment we will use the "point" approach as the primary variant and denote the "Gaussian" approach with the suffix "W". By utilizing BETULA cluster features and GMM-EM it is possible to cluster big data sets with limited memory consumption and high numerical stability, as shown in Section 6.2. Moreover, this procedure can be parallelized across multiple cores or distributed in a cluster by partitioning the data and aggregating the resulting models of all nodes (c.f. [SG18]).

6.2. Evaluation

In this section, we examine the performance of the BETULA GMM algorithms on various datasets. First, we demonstrate the numerical stability of BETULA compared to BIRCH. After that, we investigate both variants of the BETULA GMMs in terms of scaling with the dataset size. For the point GMM approaches, we then further look into the scaling with the number of leaf nodes and the dataset dimensionality. Lastly, we compare the runtimes and quality of the Gaussian Mixture Modeling algorithms on various real-world datasets and add the runtimes of the relevant k-means algorithms for comparison.

We will be using the following naming scheme: *Textbook* refers to the standard GMM implementation with the textbook variance equation $\text{Var}(X) = E[X^2] - E[X]^2$. *Stable I, D, M* denote numerically stable GMMs with isotropic (spherical), diagonal, and multivariate covariance models, respectively. *BIRCH I* refers to the BIRCH method combined with the stable isotropic model, while *BETULA I, D, M* represent BETULA combined with the stable isotropic, diagonal, and multivariate models, using the corresponding cluster features. BETULA and BIRCH models marked with the suffix *W* use the "Gaussian" likelihood function described in Eq. 6.1; otherwise, the point likelihood function is used. We use the BETULA and BIRCH algorithms with the leaves initialization. For both, we use the D4 distance criterion for distance and absorption, and 10 000 leaves with a branching factor of 48.

6.2.1. Numerical Stability

First, we examine the numerical stability of BETULA algorithms using the Correlation dataset, as described in Subsection 2.12. From a naive perspective, this dataset is easier to cluster as the distance between clusters increases until numerical stability becomes a problem. Fig. 6.1 shows the impact of numerical instabilities on the average log-likelihood as the relative cluster distance grows.

The textbook GMM algorithm remains stable until a relative distance of around $4 \cdot 10^6$, at which point instability sets in and the log-likelihood deteriorates. For BIRCH, the

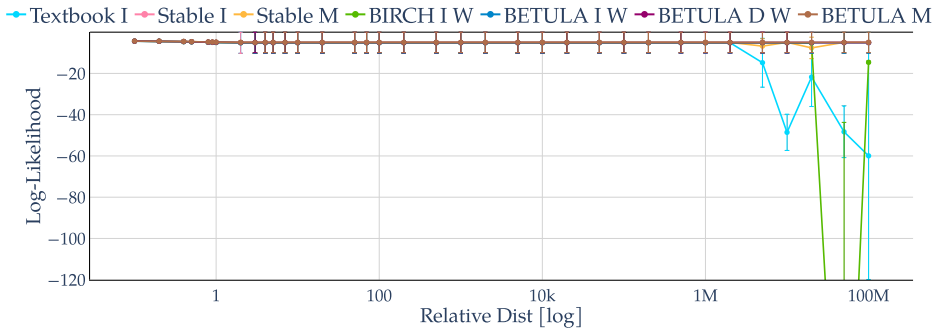
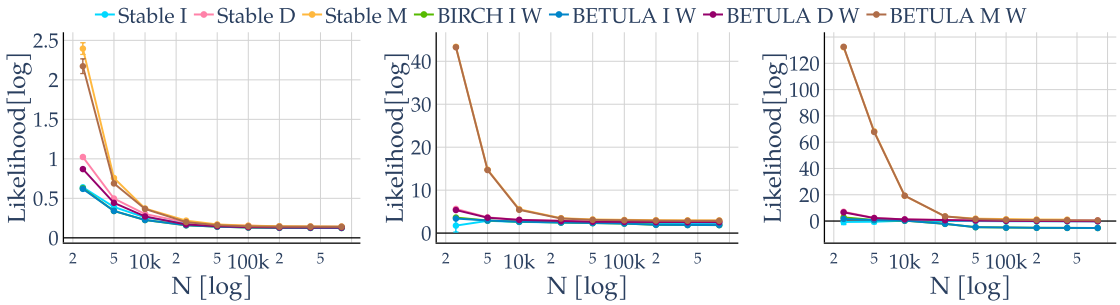


Figure 6.1.: The average log-likelihood goodness-of-fit for GMM algorithms for increasing distances between clusters as test for numerical stability.



(a) 2D (b) 8D (c) 16D

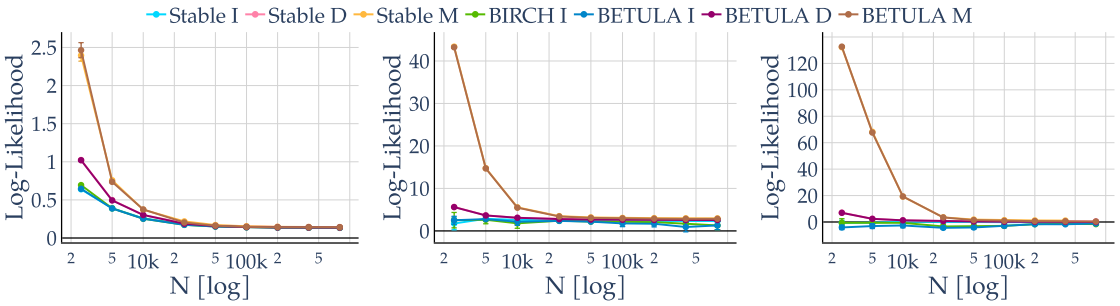
Figure 6.2.: The average log-likelihood goodness-of-fit for the Gaussian BETULA GMM algorithms for different dataset sizes and dimensionalities on the Gaussian dataset.

aggregation in cluster features distributes error more broadly, delaying instability until about $5 \cdot 10^7$, but once reached, the log-likelihood collapses dramatically (dropping out of scale to around -500), producing sudden and unpredictable results.

A deterioration at 10^7 is to be expected from double-precision calculations because of the squared values. With single-precision floating-point, it is to be expected to occur at a separation of 10^3. Both the regular GMM and BETULA are not affected and solve this idealized toy problem reliably without difficulties.

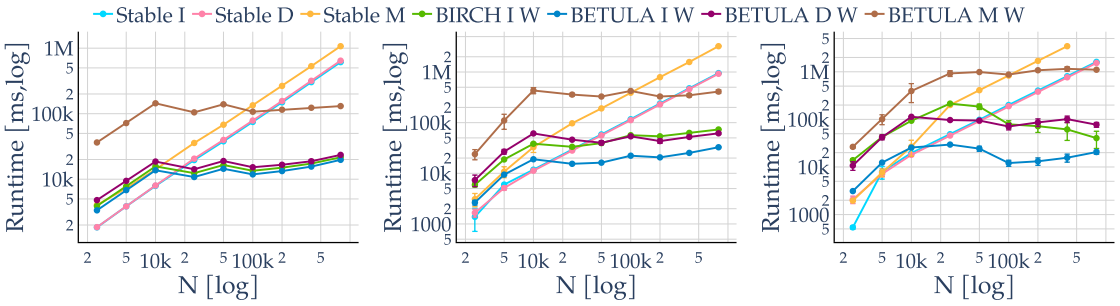
6.2.2. Quality of Gaussian Mixture Modeling

Let us now consider the average log-likelihood goodness-of-fit of the "Gaussian" BETULA GMM algorithms. As shown in Fig. 6.2a for the 2 dimensional Gaussian dataset, the difference in likelihood for bigger dataset sizes is quite minimal but for small values of N, the non-approximative stable algorithms do produce a slightly higher log-likelihood. Only the multivariate models have any change in the result values, showing very consistent results over all runs. When looking at slightly higher dimensional data,



(a) 2D (b) 8D (c) 16D

Figure 6.3.: The average log-likelihood goodness-of-fit for the point BETULA GMM algorithms for different dataset sizes and dimensionalities on the Gaussian dataset.



(a) 2D (b) 8D (c) 16D

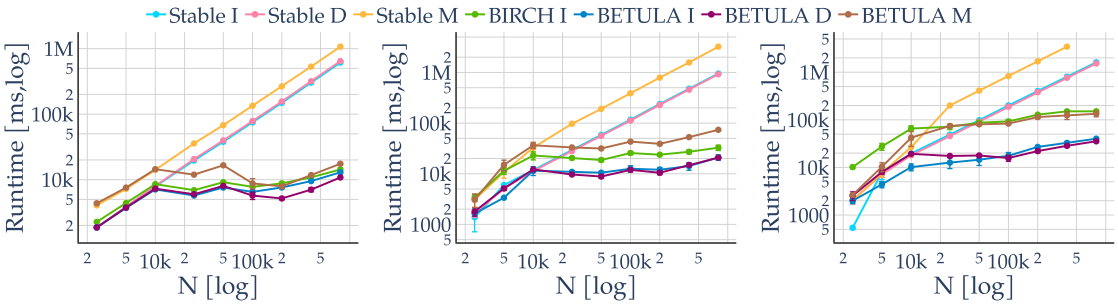
Figure 6.4.: The runtime for the Gaussian BETULA GMM algorithms for different dataset sizes and dimensionalities on the Gaussian dataset.

as shown in Fig. 6.2b and Fig. 6.2c, the difference for the multivariate models almost vanishes, and the differences for small datasets are diminishing. Instead, there is a slight difference in the likelihood for bigger N.

Turning to the point-based GMM algorithms, Fig. 6.3 shows the average log-likelihood on the Gaussian dataset across different dimensionalities. For the 2D data in Fig. 6.3a the differences between approximations and the corresponding stable exact versions are minimal. At higher dimensionality, as in Fig. 6.3c, the diagonal and multivariate models remain very close to the exact versions, while the spherical approximation shows a slight loss in quality, particularly for dataset sizes between 10 000 and 200 000. The quality of the approximations is comparable to the "Gaussian" variants.

6.2.3. Runtime Scaling with Dataset Size

Fig. 6.4 shows the runtime scaling of the BETULA variants of the "Gaussian" GMM algorithms on the Gaussian dataset with 2D, 8D, and 16D. The results indicate that all



(a) 2D (b) 8D (c) 16D

Figure 6.5.: The runtime for the point BETULA GMM algorithms for different dataset sizes and dimensionalities on the Gaussian dataset.

approximative algorithms scale very well with increasing dataset size but exhibit a significant overhead that can be seen especially for the small dataset sizes where no data aggregation takes place. This is most pronounced for the multivariate model that requires quite large datasets to compensate for this overhead and is the expected behavior due to the matrix inversions necessary; see Section 6.1.

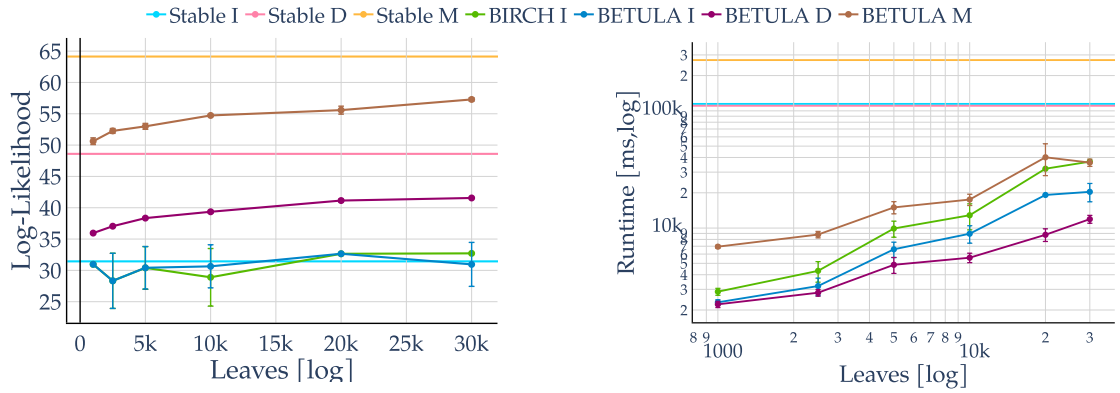
Fig. 6.5 shows the runtime scaling for the point GMM algorithms. Here, we can see only a minor overhead. With the exception of the spherical stable algorithm that, for 2500 data points on the 16D dataset, almost instantly finished. The point BETULA GMM algorithms outperform their stable equivalents significantly, even for dataset sizes where only a mild compression takes place. The runtime for the diagonal and spherical models is only minimal, making the diagonal model an interesting choice from a performance perspective.

6.2.4. Leaf Count Impact on Quality

Regarding the scaling behavior of BETULA trees on different sizes in the ALOI dataset. As shown in Fig. 6.6a, the resulting average log-likelihood varies only slightly across different leaf node counts. The BETULA spherical algorithm seems to exhibit slightly more variance in the likelihood due to variations in the initialization, which lead to different local maxima being found, but it still achieves a higher likelihood than the stable algorithm. Fig. 6.6b further demonstrates that all BETULA models achieve substantially lower runtimes compared to the standard GMM algorithms, with the diagonal model being the fastest, even outperforming the spherical variant.

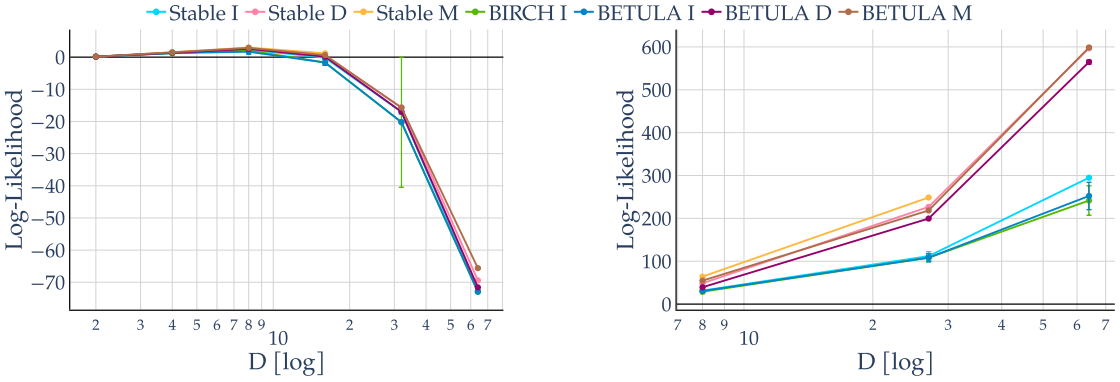
6.2.5. Scaling with Dimensionality

Next, we study the impact of increasing dimensionality on Gaussian Mixture Model average log-likelihood for our two main datasets. For the Gaussian dataset, Fig. 6.7a shows that the fit of the aggregated models is mostly indistinguishable from the corresponding stable algorithms. As dimensionality increases, the likelihood decreases,



(a) Average Log-Likelihood (b) runtime

Figure 6.6.: The runtime and average log-likelihood goodness-of-fit for GMM algorithms on the ALOI dataset with 8 dimensions for various tree sizes.



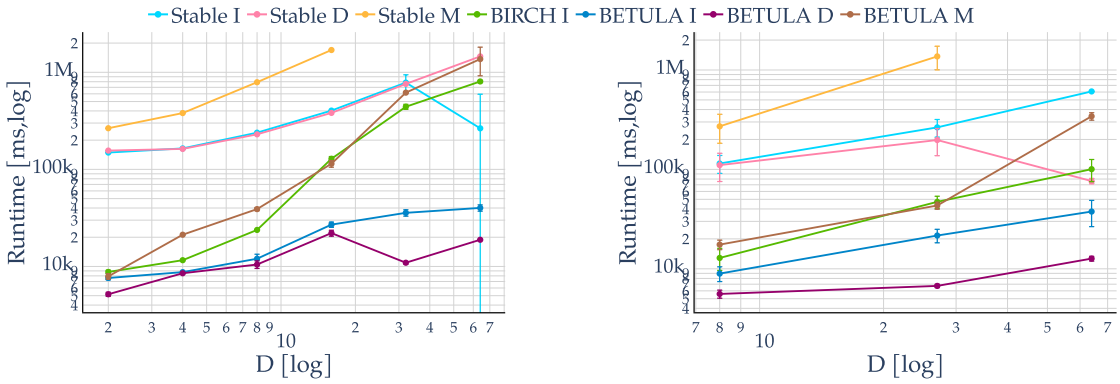
(a) Gaussian (b) ALOI

Figure 6.7.: The average log-likelihood goodness-of-fit for GMM algorithms on Gaussian and ALOI datasets with various dimensions.

which is expected since the dataset was designed such that higher-dimensional versions are progressively harder to solve. The most noticeable differences occur for BETULA and BIRCH with the spherical model, while the multivariate model stays closest to the stable baseline.

In contrast, Fig. 6.7b reports results for the ALOI dataset across different dimensionalities. Here, the 27-dimensional dataset was the largest one for which the multivariate Gaussian mixture model was able to complete within the time limit. Unlike in the synthetic case, the goodness-of-fit on the real dataset improves with dimensionality. This is because the underlying structure being modeled remains essentially the same and does not depend strongly on the number of dimensions, whereas in the Gaussian datasets we deliberately increased the difficulty as dimensionality grew.

For the spherical model, the difference between the BETULA and stable algorithms



(a) Gaussian (b) ALOI

Figure 6.8.: The runtime for GMM algorithms on Gaussian and ALOI datasets with various dimensions.

is again minimal for lower dimensionalities but becomes more pronounced starting at 64 dimensions. For the multivariate and diagonal models, the gap between aggregated and non-aggregated versions is larger but remains mostly unaffected by dimensionality.

The runtime scaling with dimensionality is shown in Fig. 6.8. As expected, the multivariate models are most sensitive to increasing dimensions due to the matrix inversions required for the log-likelihood computation, and they exceed the time limit beyond 16 dimensions. The BETULA multivariate model exhibits a similar scaling trend, but its much lower base runtime allows it to remain competitive, achieving runtimes comparable to non-multivariate stable GMMs up to 64 dimensions. For higher dimensions, however, it would eventually become slower due to scaling. BIRCH, somewhat unexpectedly, shows worse scaling than BETULA, which can be attributed to the way its summary statistics are stored; this could potentially be mitigated by converting the statistics into a format similar to BETULA after tree construction. Among all models, the spherical and diagonal BETULA variants are the fastest. Interestingly, the diagonal model is even slightly faster than the spherical one, as it appears to converge to local maxima earlier.

6.2.6. Quality and Runtime Comparison on Further Datasets

In Table 6.1, we compare the average log-likelihood of the point GMM algorithms on various datasets. For the Covertypes, KDDCUP, MNIST, and ALOI datasets, the multivariate models achieve substantially higher likelihoods than the diagonal and spherical models. For the 2-dimensional Istanbul and TrafficUK datasets, however, this is not the case. Across all datasets, the approximative BETULA variants achieve a slightly lower average log-likelihood than the stable baselines, and the loss remains consistent across all BETULA models. The BIRCH GMM variant has, as is expected with the same initialization, almost identical results as BETULA.

Next, we study the runtime behavior of the GMM algorithms on various real-world

Table 6.1.: The average log-likelihood goodness-of-fit for the GMM algorithms on various real-world datasets.

	Cover	KDDCUP	MNIST	ALOI	Istanbul	TrafficUK
Stable I	−270.286	−442.760	−47.040	112.002	−19.537	−24.755
Stable D	409.786	−249.399	55.992	226.736	−19.434	−24.724
Stable M	196.062	dnf	126.206	248.664	−19.283	dnf
BIRCH I	−275.927	−448.967	−48.344	108.251	−19.874	−24.828
BETULA I	−275.927	−448.974	−48.347	108.250	−19.877	−24.830
BETULA D	350.482	−259.732	47.888	199.549	−19.857	−24.802
BETULA M	170.510	−215.823	114.794	218.463	−19.840	−24.783

Table 6.2.: Runtime in seconds for the GMM algorithms on various real-world datasets.

	Cover	KDDCUP	MNIST	ALOI	Istanbul	TrafficUK
Stable I	2684.282	930.940	215.359	265.570	480.642	6550.597
Stable D	2683.985	944.278	239.689	198.215	501.625	6903.989
Stable M	4213.742	dnf	2121.454	1374.303	880.090	dnf
BIRCH I	126.195	209.215	108.967	47.296	12.527	80.353
BETULA I	64.257	68.790	39.016	21.756	11.296	79.606
BETULA D	63.681	18.890	40.341	6.787	11.797	85.397
BETULA M	617.796	905.341	355.427	43.594	21.819	137.233
			Model Only			
BIRCH I	99.832	200.149	106.781	44.378	7.477	14.397
BETULA I	37.763	59.706	36.813	18.835	6.218	12.702
BETULA D	35.000	9.644	37.746	3.644	6.479	13.577
BETULA M	159.978	409.223	335.007	22.723	12.743	19.668

datasets. Table 6.2 shows that, in all cases, the approximations are considerably faster than the standard GMM algorithms with the same model configuration. The BIRCH approximation is considerably more expensive than BETULA, which is because of storing the linear sum instead of the mean. This could be rectified by postprocessing the BIRCH cluster features after building the tree, which would bring them closer to BETULA cluster features but still not resolve the numerical issues. The comparatively small MNIST dataset is the only one where the stable spherical and diagonal models outperform the multivariate BETULA model. In this instance, the speedup achieved through data aggregation is the lowest, with spherical BETULA being about 5.3 times faster than the baseline, which is slightly lower than the theoretical speedup of 6, due to the relationship of the number of leaf nodes $\leq 10\,000$ and the number of entries in the dataset $(60\,000)$. The speedup for the multivariate model on this dataset is equal to 5.9, which is closest to the theoretical maximum speedup among all models and datasets.

Table 6.3.: Runtime in seconds for the a selection of k-means and GMM algorithms on various real-world datasets.

	Cover	KDDCUP	MNIST	ALOI	Istanbul	TrafficUK
Shallot	92.52	78.81	8.37	13.87	17.87	188.88
Hybrid Sha	90.35	77.56	8.53	13.10	15.86	181.71
BETULA Lloyd	17.86	11.27	4.72	3.25	2.03	17.87
Stable I	2684.28	930.94	215.36	265.57	480.64	6550.60
BETULA I	64.26	68.79	39.02	21.76	11.30	79.61
BETULA D	63.68	18.89	40.34	6.79	11.80	85.40
BETULA M	617.80	905.34	355.43	43.59	21.82	137.23

For higher compression ratios, the overhead associated with the construction of the tree remains very low, although slightly increased compared to the datasets of smaller size. In terms of models, the diagonal BETULA algorithm identifies a local maximum particularly quickly on the KDDCUP and Istanbul datasets. For the KDDCUP and TrafficUK datasets, the multivariate stable model does fails to converge within the 3 hour time limit.

When examining the runtime behavior of BETULA GMM models in more detail, it becomes apparent that in some cases the most expensive computation is the final assignment of the data to the Gaussians. Often this step is not required, as the model itself is the intended result. The runtime for computing the model, including the time for building the tree, is shown in the lower part of Table 6.2. Here, we see that the difference is again much more pronounced for the multivariate model than for the slimmer models introduced by the expensive likelihood computation for covariance-based models. Runtime without the final assignment depends mostly on the compression ratio; the higher the objects per cluster feature, the higher the runtime savings.

6.2.7. Runtime Comparison Between k-Means and GMM

Finally, the runtime of the fastest k-means algorithm is compared with the runtime of the spherical stable GMM algorithm and all BETULA-based approximations. Table 6.3 demonstrates that the BETULA k-means algorithm is, as expected, the fastest across all datasets. The multivariate model is only comparable in runtime to the non-approximative k-means algorithms on the lower-dimensional datasets, where the likelihood goodness-of-fit does not exceed that of the diagonal and spherical models. Conversely, the diagonal and spherical approximations for GMM are faster than Shallot and Hybrid Shallot on several datasets. This demonstrates a great scalability of BETULA-based approximations and establishes them as a viable alternative to k-means even for large datasets.

Chapter 7
Conclusions and Outlook

The contributions of this thesis can be seen under two complementary perspectives: enhancing the numerical stability and accelerating clustering algorithms. We present a comprehensive study and extension of the BIRCH clustering framework through the introduction of BETULA, a numerically stable refinement of the original algorithm. BETULA replaces the unstable variance computations in BIRCH with stable, running-statistics-based methods, ensuring reliable cluster feature summaries while preserving the efficiency of the CF-Tree structure. In doing so, it addresses the long-standing issue of numerical instability caused by catastrophic cancellation in variance calculations, a problem that becomes particularly severe under single-precision arithmetic. The resulting algorithm can be used in various clustering algorithms to compute high-quality approximations. In addition, this thesis introduces Cover-means, a *k*-means variant that integrates a Cover Tree to filter distance computations, thereby accelerating state-of-the-art *k*-means clustering.

Numerical Stability: To achieve numerical stability, BETULA replaces the unstable variance computation of BIRCH that uses the algebraic identity $\mathrm{Var}(X) = E[X^2] - E[X]^2$ with running statistics also used by Schubert and Gertz [SG18] and Welford [Wel62]. This enables a numerically stable variance computation while also improving the runtime in practice slightly.

Hierarchical Agglomerative Clustering (HAC): Building upon this, the thesis establishes a clear connection between cluster features and hierarchical agglomerative clustering (HAC) linkages, demonstrating how CF-Tree distances correspond to classical linkage strategies such as Ward's, centroid, and average linkage. This makes it possible to integrate CF-Trees with Anderberg's algorithm [And73] and the NN-Chain algorithm [Mur83] in a meaningful and theoretically sound way, yielding scalable hierarchical clustering for large datasets.

Gaussian Mixture Models (GMMs): BETULA cluster features can also be extended to not only provide summaries with the variance as a vector, but also store the per-dimension variance or even a correlation matrix, which can be easily converted into a

covariance matrix. This extension widens the range of clustering algorithms to be used with BETULA, enabling the integration into Gaussian mixture modeling with the EM-algorithm [DLR77]. Within Gaussian mixture models (GMMs), BETULA supports two new approaches: one based on a theoretically grounded Gaussian integral and another one based on a computationally efficient point approximation. Both variants use the numerically stable BETULA cluster features as preclustered input. The approximations for GMMs are especially useful when they compute a model of the data where no cluster assignments are required.

***k*-Means:** This thesis proposes two methods to accelerate *k*-means clustering. BETULA can be used to compute high-quality approximations very fast, and the Cover-means algorithm can be leveraged if an exact result is required. Because the initial centers for *k*-means are crucial, we propose new initialization methods, inspired by the *k*-means++ algorithm introduced by Arthur and Vassilvitskii [AV07], that work for cluster features. Those enable higher quality approximations compared to previous approximative *k*-means solutions. Finally, we contribute to the acceleration of *k*-means clustering through the presentation of the Cover-means algorithm. It leverages the Cover Tree to prune candidate centers and reduce redundant computations. Building upon this, the Hybrid Cover Tree *k*-means combines Cover Tree pruning with bound-based filtering strategies inspired by the Shallot algorithm [Bor20], resulting in measurable runtime improvements while retaining exactness. These advances establish a new state-of-the-art for tree-accelerated *k*-means.

Summary: Overall, the contributions of this thesis demonstrate how numerical stability, improved initialization, and efficient spatial partitioning can be brought together to enhance core clustering algorithms like HAC, *k*-means, and GMM alike. The results underline the efficiency of BETULA and Cover-means, ensuring that classical clustering methods remain reliable tools for modern large-scale data analysis. Moreover, with improvements to numerical stability, BETULA can also be used for edge computing devices that often have limited resources and do not necessarily support high-precision arithmetic. The reference implementations of BETULA and Cover-means are available in the ELKI framework [Sch22], making them accessible for future research and practical applications.

7.1. Future Work

There are several promising directions for extending this research. An immediate step is to systematically survey which algorithms that rely on BIRCH cluster features are affected by their numerical instability. This would help identify where the improvements introduced by BETULA could provide the greatest impact. Beyond this, implementing BETULA in a low-level language such as Rust, with Python bindings, would broaden accessibility and make it more widely usable in practical applications.

Applying BETULA's numerical stability optimizations to BICO [Fic+13] is another promising direction. Extending BICO in this way could combine the theoretical guarantees on error bounds that BICO provides with the numerical stability of BETULA. This would open the door to further applications, for example, fast cluster feature k-means initializations with guaranteed error bounds. Another application would be to investigate the possibility of introducing bounds on the approximation of Gaussian mixture models (GMMs), with the theoretical guarantees BICO can provide.

For the Cover-means approach, there are also several directions to extend the research. Further work could focus on integrating the Cover Tree even deeper into the clustering process. Replacing the current hybrid Cover Tree approach with heuristics that continue to exploit tree properties by combining the bounds of the Shallot algorithm with the Cover Tree while maintaining a lightweight bound management. This step would benefit from a more low-level implementation of Cover, which could improve on the memory management possible with Java. It would also not have the constraints of the ELKI framework, which is highly optimized, yet the modularity needed to accommodate a wide range of clustering and indexing algorithms introduces some overhead. Alternative structures, such as VP-trees, also merit investigation, as they do show competitive or superior performance for indexing. Here, asymmetric handling of near and far partitions, skipping tree levels, or selectively applying bounds may prove particularly beneficial.

Bibliography

[Agg+03] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. "A Framework for Clustering Evolving Data Streams". In: *Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003, Berlin, Germany, September 9-12, 2003*. 2003, pp. 81–92. DOI: 10.1016/B978-012722442-8/50016-1.

[Agg+04] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. "A Framework for Projected Clustering of High Dimensional Data Streams". In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004*. 2004, pp. 852–863. DOI: 10.1016/B978-012088469-8.50075-9.

[AHK01] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. "On the Surprising Behavior of Distance Metrics in High Dimensional Spaces". In: *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*. Vol. 1973. Lecture Notes in Computer Science. 2001, pp. 420–434. DOI: 10.1007/3-540-44503-X_27.

[And73] M. R. Anderberg. "Cluster Analysis for Applications". In: *Elsevier*. 1973. DOI: 10.1016/C2013-0-06161-0.

[Ank+99] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander. "OPTICS: Ordering Points To Identify the Clustering Structure". In: *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*. 1999, pp. 49–60. DOI: 10.1145/304182.304187.

[Ao+05] S. I. Ao et al. "CLUSTAG: hierarchical clustering and graph methods for selecting tag SNPs". In: *Bioinform.* 21.8 (2005), pp. 1735–1736. DOI: 10.1093/BIOINFORMATICS/BTI201.

[AV07] D. Arthur and S. Vassilvitskii. "k-means++: the advantages of careful seeding". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*. 2007, pp. 1027–1035.

[BCL09] M. Berger, M. Cole, and S. Levy. *Geometry I*. Universitext. Springer Berlin Heidelberg, 2009. ISBN: 978-3-540-11658-5.

[Ben75] J. L. Bentley. "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM* 18.9 (Sept. 1, 1975), pp. 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007.

[BFR98] P. S. Bradley, U. M. Fayyad, and C. Reina. "Scaling Clustering Algorithms to Large Databases". In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), New York City, New York, USA, August 27-31, 1998*. 1998, pp. 9–15.

[Bis07] C. M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. ISBN: 9780387310732.

[BKL06] A. Beygelzimer, S. M. Kakade, and J. Langford. "Cover trees for nearest neighbor". In: *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*. the 23rd international conference. Vol. 148. ACM International Conference Proceeding Series. Pittsburgh, Pennsylvania, 2006, pp. 97–104. ISBN: 978-1-59593-383-6. DOI: 10.1145/1143844.1143857.

[BKS00] M. M. Breunig, H. Kriegel, and J. Sander. "Fast Hierarchical Clustering Based on Compressed Data and OPTICS". In: *Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000, Proceedings*. Vol. 1910. Lecture Notes in Computer Science. 2000, pp. 232–242. DOI: 10.1007/3-540-45372-5_23.

[Bla98] J. Blackard. *Covertype* [*Dataset*]. UCI Machine Learning Repository. 1998. DOI: 10.24432/C50K5N.

[BM70] R. Bayer and E. M. McCreight. "Organization and Maintenance of Large Ordered Indices". In: *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*. SIGFIDET '70. New York, NY, USA, 1970, pp. 107–141. DOI: 10.1145/1734663.1734671.

[Boc07] H.-H. Bock. "Clustering Methods: A History of k-Means Algorithms". In: *Selected Contributions in Data Analysis and Classification*. Berlin, Heidelberg, 2007, pp. 161–172. ISBN: 978-3-540-73560-1. DOI: 10.1007/978-3-540-73560-1_15.

[Bon64] R. E. Bonner. "On Some Clustering Techniques". In: *IBM J. Res. Dev.* 8.1 (1964), pp. 22–32. DOI: 10.1147/RD.81.0022.

[Bor20] C. Borgelt. "Even Faster Exact k-Means Clustering". In: *Advances in Intelligent Data Analysis - 18th International Symposium on Intelligent Data Analysis*. Vol. 12080. Lecture Notes in Computer Science. Cham, 2020, pp. 93–105. ISBN: 978-3-030-44584-3. DOI: 10.1007/978-3-030-44584-3_8.

[Bre+01] M. M. Breunig, H. Kriegel, P. Kröger, and J. Sander. "Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering". In: *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*. SIGMOD '01. Santa Bar-

bara, California, USA, May 1, 2001, pp. 79–90. ISBN: 978-1-58113-332-5. DOI: 10.1145/375663.375672.

[Bre+84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984. ISBN: 0-534-98053-8.

[Bru77] M. Bruynooghe. "Méthodes nouvelles en classification automatique de données taxinomiques nombreuses". In: 1977.

[Cao+06] F. Cao, M. Ester, W. Qian, and A. Zhou. "Density-Based Clustering over an Evolving Data Stream with Noise". In: *Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA*. 2006, pp. 328–339. DOI: 10.1137/1.9781611972764.29.

[CBK09] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly detection: A survey". In: *ACM Comput. Surv.* 41.3 (2009), 15:1–15:58. DOI: 10.1145/1541880.1541882.

[Chi+01] T. Chiu, D. Fang, J. Chen, Y. Wang, and C. Jeris. "A robust and scalable clustering algorithm for mixed type attributes in large database environment". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*. 2001, pp. 263–268. DOI: 10.1145/502512.502549.

[CJB04] R. Caruana, T. Joachims, and L. Backstrom. "KDD-Cup 2004: results and analysis". In: *SIGKDD Explor.* 6.2 (2004), pp. 95–108. DOI: 10.1145/1046456.1046470.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces". In: *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*. 1997, pp. 426–435.

[Den12] L. Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]". In: *IEEE Signal Process. Mag.* 29.6 (2012), pp. 141–142. DOI: 10.1109/MSP.2012.2211477.

[Din+15] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, and T. Mytkowicz. "Yinyang K-Means: A Drop-In Replacement of the Classic K-Means with Consistent Speedup". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Vol. 37. JMLR Workshop and Conference Proceedings. 2015, pp. 579–587.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (Sept. 1977). Publisher: [Royal Statistical Society, Wiley], pp. 1–38. ISSN: 1467-9868. DOI: 10.1111/j.2517-6161.1977.tb01600.x.

[Elk03] C. Elkan. "Using the Triangle Inequality to Accelerate k-Means". In: *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*. 2003, pp. 147–153.

[Est+96] M. Ester, H. Kriegel, J. Sander, and X. Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*. 1996, pp. 226–231.

[Est02] V. Estivill-Castro. "Why so many clustering algorithms: a position paper". In: *SIGKDD Explor.* 4.1 (2002), pp. 65–75. DOI: 10.1145/568574.568575.

[Eve+11] B. S. Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster Analysis*. Wiley, Jan. 2011. ISBN: 9780470977811. DOI: 10.1002/9780470977811.

[Fic+13] H. Fichtenberger, M. Gillé, M. Schmidt, C. Schwiegelshohn, and C. Sohler. "BICO: BIRCH Meets Coresets for k-Means Clustering". In: *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*. Vol. 8125. Lecture Notes in Computer Science. Berlin, Heidelberg, 2013, pp. 481–492. ISBN: 978-3-642-40450-4. DOI: 10.1007/978-3-642-40450-4_41.

[For65] E. W. Forgy. "Cluster analysis of multivariate data : efficiency versus interpretability of classifications". In: *Biometrics* 21 (1965), pp. 768–769.

[FR07] C. Fraley and A. E. Raftery. "Bayesian Regularization for Normal Mixture Estimation and Model-Based Clustering". In: *J. Classif.* 24.2 (2007), pp. 155–181. DOI: 10.1007/S00357-007-0004-5.

[GGR01] V. Ganti, J. Gehrke, and R. Ramakrishnan. "DEMON: Mining and Monitoring Evolving Data". In: *IEEE Transactions on Knowledge and Data Engineering* 13.1 (Jan. 2001). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 50–63. ISSN: 2326-3865. DOI: 10.1109/69.908980.

[HA85] L. Hubert and P. Arabie. "Comparing partitions". In: *Journal of Classification* 2.1 (Dec. 1, 1985), pp. 193–218. ISSN: 1432-1343. DOI: 10.1007/BF01908075.

[Ham10] G. Hamerly. "Making k-means Even Faster". In: *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*. 0 vols. Proceedings. Apr. 29, 2010, pp. 130–140. ISBN: 978-0-89871-703-7. DOI: 10.1137/1.9781611972801.12.

[Her+16] D. Herr, Q. Han, S. Lohmann, and T. Ertl. "Visual Clutter Reduction through Hierarchy-based Projection of High-dimensional Labeled Data". In: *Proceedings of the 42nd Graphics Interface Conference, Victoria, BC, Canada, 1-3 June 2016*. 2016, pp. 109–116. DOI: 10.20380/GI2016.14.

[HKP11] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann, 2011. ISBN: 978-0123814791.

[IS15] M. Izbicki and C. Shelton. "Faster cover trees". In: International Conference on Machine Learning. July 6, 2015.

[Ján+24] M. Jánošová, A. Lang, P. Budíková, E. Schubert, and V. Dohnal. "Advancing the PAM Algorithm to Semi-supervised k-Me-doids Clustering". In: *Similarity Search and Applications - 17th International Conference, SISAP 2024, Providence, RI, USA, November 4-6, 2024, Proceedings*. Vol. 15268. Lecture Notes in Computer Science. 2024, pp. 223–237. DOI: 10.1007/978-3-031-75823-2_19.

[Ján+26] M. Jánošová, A. Lang, P. Budikova, E. Schubert, and V. Dohnal. "On the evaluation and optimization of LabeledPAM". In: *Information Systems* 135 (2026), p. 102580. ISSN: 0306-4379. DOI: 10.1016/j.is.2025.102580.

[Jin+16] C. Jin, Y. Zhang, S. Balakrishnan, M. J. Wainwright, and M. I. Jordan. "Local Maxima in the Likelihood of Gaussian Mixture Models: Structural Results and Algorithmic Consequences". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016, pp. 4116–4124.

[Jol86] I. T. Jolliffe. *Principal Component Analysis*. Springer Ser. Stat. Springer, 1986. ISBN: 978-1-4757-1906-2. DOI: 10.1007/978-1-4757-1904-8.

[Kan+02] T. Kanungo et al. "An Efficient k-Means Clustering Algorithm: Analysis and Implementation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.7 (July 2002), pp. 881–892. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2002.1017616.

[KR90] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990. ISBN: 978-0-47187876-6. DOI: 10.1002/9780470316801.

[Kra+11] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. "The ClusTree: indexing micro-clusters for anytime stream mining". In: *Knowl. Inf. Syst.* 29.2 (2011), pp. 249–272. DOI: 10.1007/S10115-010-0342-8.

[KSZ17] H. Kriegel, E. Schubert, and A. Zimek. "The (black) art of runtime evaluation: Are we comparing algorithms or implementations?" In: *Knowl. Inf. Syst.* 52.2 (2017), pp. 341–378. DOI: 10.1007/S10115-016-1004-2.

[Lap+21] M. Lapegna, W. Balzano, N. Meyer, and D. Romano. "Clustering Algorithms on Low-Power and High-Performance Devices for Edge Computing Environments". In: *Sensors* 21.16 (2021), p. 5395. DOI: 10.3390/S21165395.

[Llo57] S. P. Lloyd. "Least squares quantization in PCM". In: *IEEE Trans. Inf. Theory* 28.2 (1957), pp. 129–136. DOI: 10.1109/TIT.1982.1056489.

[LS20] A. Lang and E. Schubert. "BETULA: Numerically Stable CF-Trees for BIRCH Clustering". In: *Similarity Search and Applications - 13th International Conference, SISAP 2020, Copenhagen, Denmark, September 30 - October 2, 2020, Proceedings*. Vol. 12440. Lecture Notes in Computer Science. Cham, 2020, pp. 281–296. ISBN: 978-3-030-60936-8. DOI: 10.1007/978-3-030-60936-8_22.

[LS21] A. Lang and E. Schubert. "BETULA: Fast clustering of large data with improved BIRCH CF-Trees". In: *Information Systems* 108 (2021), p. 101918. ISSN: 0306-4379. DOI: 10.1016/J.IS.2021.101918.

[LS23] A. Lang and E. Schubert. "Accelerating k-Means Clustering with Cover Trees". In: *Similarity Search and Applications - 16th International Conference, SISAP 2023, A Coruña, Spain, October 9-11, 2023, Proceedings*. Vol. 14289. Lecture Notes in Computer Science. 2023, pp. 148–162. DOI: 10.1007/978-3-031-46994-7_13.

[LW66] G. N. Lance and W. T. Williams. "A Generalized Sorting Strategy for Computer Classifications". In: *Nature* 212.5058 (Oct. 1966), pp. 218–218. ISSN: 1476-4687. DOI: 10.1038/212218a0.

[LW67] G. N. Lance and W. T. Williams. "A General Theory of Classificatory Sorting Strategies: 1. Hierarchical Systems". In: *Comput. J.* 9.4 (1967), pp. 373–380. DOI: 10.1093/COMJNL/9.4.373.

[MC12] F. Murtagh and P. Contreras. "Algorithms for hierarchical clustering: an overview". In: *WIREs Data Mining Knowl. Discov.* 2.1 (2012), pp. 86–97. DOI: 10.1002/WIDM.53.

[McQ57] L. L. McQuitty. "Elementary Linkage Analysis for Isolating Orthogonal and Oblique Types and Typal Relevancies". In: *Educational and Psychological Measurement* 17.2 (1957), pp. 207–229. DOI: 10.1177/001316445701700204.

[MKE16] S. Miyamoto, Y. Kaizu, and Y. Endo. "Hierarchical and Non-Hierarchical Medoid Clustering Using Asymmetric Similarity Measures". In: *2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS), Sapporo, Japan, August 25-28, 2016*. 2016, pp. 400–403. DOI: 10.1109/SCIS-ISIS.2016.0091.

[ML14] F. Murtagh and P. Legendre. "Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion?" In: *J. Classif.* 31.3 (2014), pp. 274–295. DOI: 10.1007/S00357-014-9161-Z.

[MNV09] M. Mahajan, P. Nimbhorkar, and K. R. Varadarajan. "The Planar k-Means Problem is NP-Hard". In: *WALCOM: Algorithms and Computation, Third International Workshop, WALCOM 2009, Kolkata, India, February 18-20, 2009. Proceedings*. Vol. 5431. Lecture Notes in Computer Science. 2009, pp. 274–285. DOI: 10.1007/978-3-642-00202-1_24.

[Mur83] F. Murtagh. "A Survey of Recent Advances in Hierarchical Clustering Algorithms". In: *Comput. J.* 26.4 (1983), pp. 354–359. DOI: 10.1093/COMJNL/26.4.354.

[NF16] J. Newling and F. Fleuret. "Fast k-means with accurate bounds". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Vol. 48. JMLR Workshop and Conference Proceedings. 2016, pp. 936–944.

[Ost+06] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy. "The Effectiveness of Lloyd-Type Methods for the k-Means Problem". In: *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*. 2006, pp. 165–176. DOI: 10.1109/FOCS.2006.75.

[Phi02] S. J. Phillips. "Acceleration of K-Means and Related Clustering Algorithms". In: *Algorithm Engineering and Experiments, 4th International Workshop, ALENEX*. Vol. 2409. Lecture Notes in Computer Science. Berlin, Heidelberg, 2002, pp. 166–177. ISBN: 978-3-540-45643-8. DOI: 10.1007/3-540-45643-0_13.

[PM99] D. Pelleg and A. W. Moore. "Accelerating Exact *k*-means Algorithms with Geometric Reasoning". In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999*. San Diego, California, United States, 1999, pp. 277–281. ISBN: 978-1-58113-143-7. DOI: 10.1145/312129.312248.

[Pod89] J. P. Podani. "New combinatorial clustering methods". In: *Vegetatio* 81.1–2 (July 1989). Publisher: Springer, pp. 61–77. ISSN: 1573-5052. DOI: 10.1007/BF00045513.

[Rab89] L. R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proc. IEEE* 77.2 (1989), pp. 257–286. DOI: 10.1109/5.18626.

[RQD00] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn. "Speaker Verification Using Adapted Gaussian Mixture Models". In: *Digit. Signal Process.* 10.1-3 (2000), pp. 19–41. DOI: 10.1006/DSPR.1999.0361.

[Sch21a] E. Schubert. "A Triangle Inequality for Cosine Similarity". In: *Similarity Search and Applications - 14th International Conference, SISAP 2021, Dortmund, Germany, September 29 - October 1, 2021, Proceedings*. Vol. 13058. Lecture Notes in Computer Science. 2021, pp. 32–44. DOI: 10.1007/978-3-030-89657-7_3.

[Sch21b] E. Schubert. "HACAM: Hierarchical Agglomerative Clustering Around Medoids - and its Limitations". In: *Proceedings of the LWDA 2021 Workshops: FGWM, KDML, FGWI-BIA, and FGIR, Online, September 1-3, 2021*. Vol. 2993. CEUR Workshop Proceedings. 2021, pp. 191–204.

[Sch22] E. Schubert. "Automatic Indexing for Similarity Search in ELKI". In: *Similarity Search and Applications - 15th International Conference, SISAP 2022, Bologna, Italy, October 5-7, 2022, Proceedings*. Vol. 13590. Lecture Notes in Computer Science. 2022, pp. 205–213. DOI: 10.1007/978-3-031-17849-8_16.

[Sch24] E. Schubert. "Hierarchical Clustering Without Pairwise Distances by Incremental Similarity Search". In: *Similarity Search and Applications - 17th International Conference, SISAP 2024, Providence, RI, USA, November 4-6, 2024, Proceedings*. Vol. 15268. Lecture Notes in Computer Science. 2024, pp. 238–252. ISBN: 9783031758232. DOI: 10.1007/978-3-031-75823-2_20.

[Scu10] D. Sculley. "Web-scale k-means clustering". In: *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*. Raleigh, North Carolina, USA, 2010, pp. 1177–1178. ISBN: 978-1-60558-799-8. DOI: 10.1145/1772690.1772862.

[SG18] E. Schubert and M. Gertz. "Numerically Stable Parallel Computation of (Co-)Variance". In: 2018. DOI: 10.1145/3221269.3223036.

[SI84] S. Z. Selim and M. A. Ismail. "K-Means-Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 6.1 (1984), pp. 81–87. DOI: 10.1109/TPAMI.1984.4767478.

[SKP15] F. Schroff, D. Kalenichenko, and J. Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 2015, pp. 815–823. DOI: 10.1109/CVPR.2015.7298682.

[SL22] E. Schubert and A. Lang. "Data Aggregation for Hierarchical Clustering". In: *Machine Learning under Resource Constraints - Volume 1: Fundamentals*. De Gruyter STEM. 2022, pp. 215–226. DOI: 10.1515/9783110785944-005.

[SLF21] E. Schubert, A. Lang, and G. Feher. "Accelerating Spherical k-Means". In: *Similarity Search and Applications - 14th International Conference, SISAP 2021, Dortmund, Germany, September 29 - October 1, 2021, Proceedings*. Vol. 13058. Lecture Notes in Computer Science. 2021, pp. 217–231. DOI: 10.1007/978-3-030-89657-7_17.

[Sne57] P. H. A. Sneath. "The Application of Computers to Taxonomy". In: *Microbiology* 17.1 (Aug. 1957), pp. 201–226. ISSN: 1465-2080. DOI: 10.1099/00221287-17-1-201.

[SR21] E. Schubert and P. J. Rousseeuw. "Fast and Eager k-Medoids Clustering: O(k) Runtime Improvement of the PAM, CLARA, and CLARANS Algorithms". In: *Information Systems* 101 (Nov. 2021), p. 101804. ISSN: 0306-4379. DOI: 10.1016/j.is.2021.101804. arXiv: 2008.05171.

[SR62] R. R. Sokal and F. J. Rohlf. "The Comparison of Dendrograms by Objective Methods". In: *Taxon* 11.2 (1962), pp. 33–40. ISSN: 00400262.

[SS63] P. H. Sokal and R. R. Sneath. *Principles of Numerical Taxonomy. Books in biology*. W.H. Freeman, 1963.

[Ste57] H. Steinhaus. "Sur la division des corps matériels en parties". In: *Bulletin L'Académie Polonaise des Science* 4.12 (1957), pp. 801–804.

[SZ10] E. Schubert and A. Zimek. *ELKI Multi-View Clustering Data Sets Based on the Amsterdam Library of Object Images (ALOI)*. 2010. DOI: 10.5281/ZENODO.6355684.

[Wel62] B. P. Welford. "Note on a Method for Calculating Corrected Sums of Squares and Products". In: *Technometrics* 4.3 (Aug. 1962), pp. 419–420. ISSN: 1537-2723. DOI: 10.1080/00401706.1962.10490022.

[Wis69] D. Wishart. "256. Note: An Algorithm for Hierarchical Classifications". In: *Biometrics* 25.1 (1969), pp. 165–170. ISSN: 0006-341X. DOI: 10.2307/2528688.

[YCC20] Q. Yu, K. Chen, and J. Chen. "Using a Set of Triangle Inequalities to Accelerate K-means Clustering". In: *Similarity Search and Applications - 13th International Conference, SISAP 2020, Copenhagen, Denmark, September 30 - October 2, 2020, Proceedings*. Vol. 12440. Lecture Notes in Computer Science. Cham, 2020, pp. 297–311. ISBN: 978-3-030-60936-8. DOI: 10.1007/978-3-030-60936-8_23.

[Yia93] P. N. Yianilos. "Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces". In: *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*. SODA '93. USA, Jan. 1, 1993, pp. 311–321. ISBN: 978-0-89871-313-8.

[Zez+06] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search - The Metric Space Approach*. Vol. 32. Advances in Database Systems. Kluwer, 2006. ISBN: 978-0-387-29146-8. DOI: 10.1007/0-387-29151-2.

[Zha96] T. Zhang. *Data Clustering for Very Large Datasets Plus Applications*. Tech. rep. 1355. University of Wisconsin Madison, 1996.

[Ziv04] Z. Zivkovic. "Improved Adaptive Gaussian Mixture Model for Background Subtraction". In: *17th International Conference on Pattern Recognition, ICPR 2004, Cambridge, UK, August 23-26, 2004*. 2004, pp. 28–31. DOI: 10.1109/ICPR.2004.1333992.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: An Efficient Data Clustering Method for Very Large Databases". In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*. 1996, pp. 103–114. DOI: 10.1145/233269.233324.

[ZRL97] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: A New Data Clustering Algorithm and Its Applications". In: *Data Mining and Knowledge Discovery* 1.2 (June 1, 1997), pp. 141–182. ISSN: 1573-756X. DOI: 10.1023/A:1009783824328.

[ZRL99] T. Zhang, R. Ramakrishnan, and M. Livny. "Fast Density Estimation Using CF-Kernel for Very Large Databases". In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999*. 1999, pp. 312–316. DOI: 10.1145/312129.312266.

Chapter A
Appendix

A.1. Derivation of the BIRCH Distance Criteria

Used equalities for substitutions

$$\frac{1}{n}\sum_{i=1}^{n}\|x-\mu\|^2 = \mathrm{Var}(X) = E(X^2) - E(X)^2 = \frac{1}{n}SS - \|\frac{\overrightarrow{LS}}{n}\|^2$$

$$\begin{aligned}
\mathrm{D2}(A,B)^2 &= \frac{1}{n_A n_B}\sum_{x\in A}\sum_{y\in B}\|x-y\|^2 \\
&= \frac{1}{2n_A n_B}\Big(\sum_{x,y\in AB}\|x-y\|^2 - \sum_{x,y\in A}\|x-y\|^2 - \sum_{x,y\in B}\|x-y\|^2\Big) \\
&= \frac{1}{n_A n_B}\Big(n_{AB}\sum_{x\in AB}\|x-\mu\|^2 - n_A\sum_{x\in A}\|x-\mu_A\|^2 - n_B\sum_{x\in B}\|x-\mu_B\|^2\Big) \\
&= \frac{1}{n_A n_B}\Big((n_{AB}SS_{AB} - \|\overrightarrow{LS}_{AB}\|^2) - (n_A SS_A - \|\overrightarrow{LS}_A\|^2) - (n_B SS_B - \|\overrightarrow{LS}_B\|^2)\Big) \\
&= \frac{1}{n_A n_B}\Big(n_B SS_A + n_A SS_B + (\|\overrightarrow{LS}_A\|^2 + \|\overrightarrow{LS}_B\|^2 - \|\overrightarrow{LS}_A + \overrightarrow{LS}_B\|^2)\Big) \\
&= \frac{1}{n_A n_B}(n_B SS_A + n_A SS_B - 2\overrightarrow{LS}_A^T\overrightarrow{LS}_B) \\
\mathrm{D3}(A,B)^2 &= \frac{1}{n_{AB}(n_{AB}-1)}\sum_{x,y\in AB}\|x-y\|^2 = \frac{2n_{AB}}{n_{AB}(n_{AB}-1)}\sum_{x\in AB}\|x-\mu\|^2 \\
&= \frac{2}{n_{AB}-1}\Big(SS_{AB} - n_{AB}\|\frac{\overrightarrow{LS}_{AB}}{n_{AB}}\|^2\Big) \\
&= \frac{2}{n_A+n_B-1}(SS_A + SS_B - \frac{1}{n_A+n_B}\|\overrightarrow{LS}_A+\overrightarrow{LS}_B\|^2) \\
\mathrm{D4}(A,B)^2 &= \sum_{x\in AB}\|x-\mu_{AB}\|^2 - \sum_{x\in A}\|x-\mu_A\|^2 - \sum_{x\in B}\|x-\mu_B\|^2 \\
&= SS_{AB} - \frac{1}{n_{AB}}\|\overrightarrow{LS}_{AB}\|^2 - \Big(SS_A - \frac{1}{n_A}\|\overrightarrow{LS}_A\|^2\Big) - \Big(SS_B - \frac{1}{n_B}\|\overrightarrow{LS}_B\|^2\Big) \\
&= SS_{AB} - SS_A - SS_B + \frac{1}{n_A}\|\overrightarrow{LS}_A\|^2 + \frac{1}{n_B}\|\overrightarrow{LS}_B\|^2 - \frac{1}{n_{AB}}\|\overrightarrow{LS}_{AB}\|^2 \\
&= \frac{1}{n_A}\|\overrightarrow{LS}_A\|^2 + \frac{1}{n_B}\|\overrightarrow{LS}_B\|^2 - \frac{1}{n_A+n_B}\|\overrightarrow{LS}_A+\overrightarrow{LS}_B\|^2 \\
\mathrm{R}(A,B)^2 &= \frac{1}{n_{AB}}\sum_{x\in AB}\|x-\mu_{AB}\|^2 = \frac{1}{n_{AB}}SS_{AB} - \|\frac{\overrightarrow{LS}_{AB}}{n_{AB}}\|^2 \\
&= \frac{1}{n_{AB}}(SS_{AB} - \frac{1}{n_{AB}}\|\overrightarrow{LS}_{AB}\|^2)
\end{aligned}$$

A.2. Derivation of the BETULA Distance Criteria

Used equalities for substitutions.

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\|x_i-x_j\|^2 = 2n\sum_{i=1}^{n}\|x_i-\mu\|^2$$
$$S_{AB} = S_A + S_B + \tfrac{n_A n_B}{n_A+n_B}\|\mu_A-\mu_B\|^2$$

$$\begin{aligned}
\mathrm{D2}(A,B)^2 &= \tfrac{1}{n_A n_B}\sum_{x\in A}\sum_{y\in B}\|x-y\|^2 \\
&= \tfrac{1}{2n_A n_B}\Big(\sum_{x,y\in AB}\|x-y\|^2 - \sum_{x,y\in A}\|x-y\|^2 - \sum_{x,y\in B}\|x-y\|^2\Big) \\
&= \tfrac{1}{n_A n_B}\Big(n_{AB}\sum_{x\in AB}\|x-\mu\|^2 - n_A\sum_{x\in A}\|x-\mu_A\|^2 - n_B\sum_{x\in B}\|x-\mu_B\|^2\Big) \\
&= \tfrac{1}{n_A n_B}\left(n_{AB}S_{AB} - n_A S_A - n_B S_B\right) \\
&= \tfrac{1}{n_A n_B}\left(n_{AB}\left(S_A + S_B + \tfrac{n_A n_B}{n_A+n_B}\|\mu_A-\mu_B\|^2\right) - n_A S_A - n_B S_B\right) \\
&= \tfrac{n_B}{n_A n_B}S_A + \tfrac{n_A}{n_A n_B}S_B + \tfrac{n_A n_B}{n_A n_B}\|\mu_A-\mu_B\|^2 \\
&= \tfrac{1}{n_A}S_A + \tfrac{1}{n_B}S_B + \|\mu_A-\mu_B\|^2 \\
\mathrm{D3}(A,B)^2 &= \tfrac{1}{n_{AB}(n_{AB}-1)}\sum_{x,y\in AB}\|x-y\|^2 = \tfrac{2}{n_{AB}(n_{AB}-1)}n_{AB}\sum_{x\in AB}\|x-\mu\|^2 \\
&= \tfrac{2}{n_{AB}(n_{AB}-1)}n_{AB}S_{AB} \\
&= \tfrac{2}{n_{AB}(n_{AB}-1)}n_{AB}\left(S_A + S_B + \tfrac{n_A n_B}{n_{AB}}\|\mu_A-\mu_B\|^2\right) \\
&= \tfrac{2}{n_{AB}(n_{AB}-1)}(n_{AB}(S_A+S_B) + n_A n_B\|\mu_A-\mu_B\|^2) \\
\mathrm{D4}(A,B)^2 &= \sum_{x\in AB}\|x-\mu_{AB}\|^2 - \sum_{x\in A}\|x-\mu_A\|^2 - \sum_{x\in B}\|x-\mu_B\|^2 \\
&= S_{AB} - S_A - S_B = S_A + S_B + \tfrac{n_A n_B}{n_{AB}}\|\mu_A-\mu_B\|^2 - S_A - S_B \\
&= \tfrac{n_A n_B}{n_{AB}}\|\mu_A-\mu_B\|^2 \\
\mathrm{R}(A,B)^2 &= \tfrac{1}{n_{AB}}\sum_{x\in AB}\|x-\mu_{AB}\|^2 = \tfrac{1}{n_{AB}}S_{AB} \\
&= \tfrac{1}{n_{AB}}(S_A + S_B + \tfrac{n_A\cdot n_B}{n_{AB}}\|\mu_A-\mu_B\|^2)
\end{aligned}$$

A.3. Derivation of the BETULA Linkage criteria

Ward:

$$
\begin{aligned}
d_{\text{Ward}}(A,B) =& \sum_{x\in A\cup B} \|x-\mu_{AB}\|^2 - \sum_{a\in A} \|a-\mu_A\|^2 - \sum_{b\in B} \|b-\mu_B\|^2 \\
=& \sum_{a\in A} \left(\|a-\mu_{AB}\|^2 - \|a-\mu_A\|^2 \right) \\
&+ \sum_{b\in B} \left(\|b-\mu_{AB}\|^2 - \|b-\mu_B\|^2 \right) \\
=& \Big(\sum_{a\in A} (a-\mu_{AB} + a - \mu_A) \Big)(\mu_A - \mu_{AB}) \\
&+ \Big(\sum_{b\in B} (b-\mu_{AB} + b - \mu_B) \Big)(\mu_B - \mu_{AB}) \\
=& n_A(2\mu_A - \mu_{AB} - \mu_A)(\mu_A - \mu_{AB}) \\
&+ n_B(2\mu_B - \mu_{AB} - \mu_B)(\mu_B - \mu_{AB}) \\
=& n_A \|\mu_A - \mu_{AB}\|^2 + n_B \|\mu_B - \mu_{AB}\|^2 \\
=& n_A \left\|\mu_A - \tfrac{n_A}{n_{AB}}\mu_A - \tfrac{n_B}{n_{AB}}\mu_B\right\|^2 + n_B \left\|\mu_B - \tfrac{n_A}{n_{AB}}\mu_A - \tfrac{n_B}{n_{AB}}\mu_B\right\|^2 \\
=& \tfrac{2n_An_B}{n_{AB}} \|\mu_A - \mu_B\|^2
\end{aligned}
$$

$$
\begin{aligned}
d_{\text{Ward}}(A\cup B,C) =& \tfrac{2n_{AB}n_C}{n_{ABC}} \|\mu_{AB} - \mu_C\|^2 \\
=& \tfrac{2n_{AB}n_C}{n_{ABC}} \left\|\tfrac{n_A}{n_{AB}}\mu_A + \tfrac{n_B}{n_{AB}}\mu_B - \mu_C\right\|^2 \\
=& \tfrac{2n_C}{n_{ABC}n_{AB}} \|n_A(\mu_A - \mu_C) + n_B(\mu_B - \mu_C)\|^2 \\
=& \tfrac{2n_C}{n_{ABC}n_{AB}} \Big(n_A^2 \|\mu_A - \mu_C\|^2 + n_B^2 \|\mu_B - \mu_C\|^2 \\
&+ 2n_An_B(\mu_A - \mu_C)(\mu_B - \mu_C) \Big) \\
=& \tfrac{2n_C}{n_{ABC}n_{AB}} \Big(n_A^2 \|\mu_A - \mu_C\|^2 + n_B^2 \|\mu_B - \mu_C\|^2 + n_An_B \cdot \\
&(\|\mu_A - \mu_C\|^2 + \|\mu_B - \mu_C\|^2 - \|(\mu_A - \mu_C) - (\mu_B - \mu_C)\|^2) \Big) \\
=& \tfrac{2n_C}{n_{ABC}n_{AB}} (n_An_{AB} \|\mu_A - \mu_C\|^2 + n_Bn_{AB} \|\mu_B - \mu_C\|^2 \\
&- n_An_B \|\mu_A - \mu_B\|^2) \\
=& \tfrac{2n_An_C}{n_{ABC}} \|\mu_A - \mu_C\|^2 + \tfrac{2n_Bn_C}{n_{ABC}} \|\mu_B - \mu_C\|^2 \\
&- \tfrac{n_C}{n_{ABC}} \tfrac{2n_An_B}{n_{AB}} \|\mu_A - \mu_B\|^2 \\
=& \tfrac{n_{AC}}{n_{ABC}} d_{\text{Ward}}(A,C) + \tfrac{n_{BC}}{n_{ABC}} d_{\text{Ward}}(B,C) - \tfrac{n_C}{n_{ABC}} d_{\text{Ward}}(A,B)
\end{aligned}
$$

Centroid:

$$
\begin{aligned}
d_{\text{cent}}(A \cup B, C) =& \frac{n_A}{n_{AB}} d_{\text{cent}}(A, C) + \frac{n_B}{n_{AB}} d_{\text{cent}}(B, C) - \frac{n_A n_B}{{n_{AB}}^2} d_{\text{cent}}(A, B) \\
=& \frac{n_A}{n_{AB}} \|\mu_A - \mu_C\|^2 + \frac{n_B}{n_{AB}} \|\mu_B - \mu_C\|^2 - \frac{n_A n_B}{{n_{AB}}^2} \|\mu_A - \mu_B\|^2 \\
=& \frac{n_A}{n_{AB}} \mu_A^2 + \frac{n_A}{n_{AB}} \mu_C^2 - 2 \frac{n_A}{n_{AB}} \mu_A \mu_C \\
& + \frac{n_B}{n_{AB}} \mu_B^2 + \frac{n_B}{n_{AB}} \mu_C^2 - 2 \frac{n_B}{n_{AB}} \mu_B \mu_C \\
& - \frac{n_A n_B}{{n_{AB}}^2} \mu_A^2 - \frac{n_A n_B}{{n_{AB}}^2} \mu_B^2 + 2 \frac{n_A n_B}{{n_{AB}}^2} \mu_A \mu_B \\
=& \frac{n_A^2}{{n_{AB}}^2} \mu_A^2 + \frac{n_B^2}{{n_{AB}}^2} \mu_B^2 + 2 \frac{n_A n_B}{{n_{AB}}^2} \mu_A \mu_B \\
& + \mu_C^2 - 2 \left(\frac{n_A}{n_{AB}} \mu_A \mu_C + \frac{n_B}{n_{AB}} \mu_B \mu_C \right) \\
=& \left(\frac{n_A}{n_{AB}} \mu_A + \frac{n_B}{n_{AB}} \mu_B \right)^2 + \mu_C^2 - 2 \left(\frac{n_A}{n_{AB}} \mu_A + \frac{n_B}{n_{AB}} \mu_B \right) \mu_C \\
=& \mu_{AB}^2 + \mu_C^2 - 2 \mu_{AB} \mu_C = \|\mu_{AB} - \mu_C\|^2 \quad .
\end{aligned}
$$

A.4. Quality of Distance / Absorption Criteria on the ALOI Dataset

Table A.1.: RMSD after pre-clustering with distance and absorption criteria combinations, and the resulting number of leaf entries on the ALOI dataset.

	metric	RMSD / 10^{-2}		Leaves	
Dist.	Abs.	Mean	95% CI	mean	95% CI
D0	D0	1.67	±0.054	7862	±178
	D1	2.11	±0.177	5052	±1058
	D2	1.84	±0.034	5653	±173
	D3	1.51	±0.032	9670	±172
	D4	1.51	±0.027	5760	±88
	R	1.65	±0.038	7342	±159
D1	D0	1.47	±0.025	8614	±124
	D1	1.82	±0.063	5626	±187
	D2	1.59	±0.033	6710	±140
	D3	1.84	±0.060	5569	±195
	D4	1.34	±0.017	6441	±97
	R	1.40	±0.032	8655	±102
D2	D0	1.80	±0.093	7089	±206
	D1	1.57	±0.037	9156	±188
	D2	1.95	±0.153	5645	±1353
	D3	1.64	±0.031	8835	±433
	D4	1.61	±0.044	5431	±154
	R	1.83	±0.052	6260	±271
D3	D0	10.55	±0.454	6933	±220
	D1	11.32	±0.258	8172	±679
	D2	11.18	±0.383	8550	±709
	D3	10.80	±0.363	7842	±493
	D4	11.74	±0.594	8419	±315
	R	11.52	±0.344	9497	±282
D4	D0	1.30	±0.018	8297	±152
	D1	1.57	±0.064	5415	±90
	D2	1.45	±0.029	6211	±116
	D3	1.75	±0.067	5193	±154
	D4	1.27	±0.051	7407	±65
	R	1.38	±0.022	7885	±94
R	D0	10.70	±0.644	8265	±1337
	D1	11.57	±0.666	7836	±1014
	D2	11.34	±0.361	7737	±826
	D3	10.29	±0.362	4866	±578
	D4	10.93	±0.479	8766	±456
	R	10.49	±0.290	5836	±630