GO_Clustering.jl: A Unified Framework for Globally Optimal Clustering

by

Yu Wang

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Mathematics)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

April 2026

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

GO_Clustering.jl: A Unified Framework for Globally Optimal Clustering

submitted by **Yu Wang** in partial fulfillment of the requirements for the degree of **Master of Science** in **Mathematics**.

Examining Committee:

Yankai Cao, Associate Professor, Chemical and Biological Engineering, UBC
Supervisor

Kefei Wen, Assistant Professor, Department of Mechanical Engineering , UBC
Supervisory Committee Member

Abstract

This thesis studies globally optimal solution methods for three classical centroid-based clustering problems: K-means, K-centers, and K-medoids. The work has two closely related objectives. First, it presents a unified mathematical treatment of these models, including mixed-integer formulations and a common branch-and-bound viewpoint that clarifies the roles of lower bounds, upper bounds, domain reduction, and branching. Second, it implements this viewpoint in a Julia package, `GO_Clustering.jl`, that brings together exact and gap-certifying methods from the literature within a consistent computational framework. The package provides common data structures, solver interfaces, MPI-based parallel execution, structured logging, and experiment drivers, so that one software environment can be used to solve and compare K-means, K-centers, and K-medoids under a unified workflow. By integrating these methods into a single package, the thesis supports systematic computational study and makes it easier to report certified lower and upper bounds and global optimality certificates. Overall, the thesis provides a reproducible software foundation for research on globally optimal centroid-based clustering.

Lay Summary

Clustering is a way of grouping data points that are similar to one another. It is used in areas such as market analysis, biology, and image processing. Most clustering software uses fast heuristic methods, which can work well in practice but cannot guarantee the best possible grouping. This thesis studies three standard models—K-means, K-centers, and K-medoids—and develops a Julia package that brings together existing exact methods for solving them with mathematical certificates of quality. The main contribution is not the invention of new algorithms. Instead, the thesis organizes known formulations and published solution methods into one consistent software framework, making it easier to compare methods, run reproducible experiments, and study when globally optimal clustering is computationally practical.

Preface

This thesis is an original work carried out in collaboration with and under the guidance of my supervisor, Professor Yankai Cao.

The algorithmic ideas and solution methods discussed in this thesis were proposed by Professor Cao and his research team. For the software reported in this thesis, Engels Emiliano Miranda Palacios implemented the single-threaded algorithmic framework. My main contributions were in the code implementation and computational study: I implemented the parallel framework, developed part of the package functionality, conducted the numerical experiments, analyzed the results, and prepared the thesis manuscript.

Generative artificial intelligence was used in preparing this thesis. Specifically, ChatGPT was used to assist with language polishing and with preparing figures and diagrams. All research design, mathematical modelling, computational experiments, interpretation of results.

At the time of submission, no journal articles or conference papers had arisen directly from the work presented in this thesis.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Yankai Cao, for his guidance, support, and generous help throughout my research and the preparation of this thesis.

I would also like to thank Kaixun Hua, Mingfei Shi, Jiayang Ren and Engels Emiliano Miranda Palacios. Their previous research provided valuable insights and a helpful foundation for my work.

Chapter 1

Introduction

1.1 Problem Setting and Motivation

Clustering studies how to partition a dataset $\mathcal{X} = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ into k groups so that points assigned to the same group are, in an application-dependent sense, more similar to one another than to points assigned to different groups. It is one of the most established problems in machine learning, statistics, and operations research, and it underpins a wide range of scientific and engineering tasks, including customer segmentation, genomic data analysis, image quantisation, network analysis, facility design, and data compression (Jain, 2010). In many of these settings, clustering is not merely an exploratory device; it serves as an intermediate decision layer whose output influences subsequent inference, optimisation, or policy design.

Among the many clustering paradigms that have been proposed, centroid-based methods occupy a particularly important position. Their popularity stems from a combination of conceptual simplicity, computational tractability, and interpretability. K-means remains the canonical example: its classical formulation is commonly attributed to MacQueen (1967), its standard iterative implementation is due to Lloyd (Lloyd, 1982), and its practical performance has motivated a large literature on initialisation, acceleration, and approximation, including K-means++ (Arthur & Vassilvitskii, 2007). More broadly, K-means has been identified as one of the most influential algorithms in data mining (Wu et al., 2008). Closely related families such as K-medoids and K-centers replace the centroid objective with representative-sample or worst-case distance criteria, respectively, and are frequently preferred when robustness, metric generality, or worst-case guarantees are more important than squared-error fit (Gonzalez, 1985; Kaufman & Rousseeuw, 1990).

The practical success of centroid-based clustering, however, should not obscure an important methodological fact: the algorithms most often used in

practice are heuristic. For K-means, Lloyd's method is monotone but converges only to a local optimum, and the quality of the final solution depends materially on the initial seeds. K-means++ improves initialisation quality in expectation, but it does not eliminate the underlying non-convexity of the optimisation problem. Similar issues arise for K-medoids and K-centers, where widely used procedures often provide efficient approximate solutions but do not certify global optimality. In many routine applications this limitation is accepted as a practical compromise. In settings where clustering is used to build benchmarks, compare algorithms, design representative partitions, or support downstream high-stakes analysis, however, the lack of optimality guarantees becomes a substantive methodological concern rather than a minor implementation detail.

This concern is reinforced by known complexity barriers. Minimum-sum-of-squares clustering is NP-hard even in low-dimensional Euclidean settings (Aloise et al., 2009; Mahajan et al., 2012). Related hardness phenomena also arise for medoid-based and max-distance formulations (Gonzalez, 1985; Kaufman & Rousseeuw, 1990). The central question is therefore not whether globally optimal clustering is always easy, but under what mathematical structures, algorithmic designs, and computational regimes globally optimal or rigorously gap-certified solutions can be obtained at practically meaningful scales.

1.2 Exactness, Scalability, and Software Gaps

The discussion above motivates a more focused question: under what mathematical and computational conditions can centroid-based clustering be solved with certified optimality at practically relevant scales? This question gives rise to three closely related issues concerning exactness, scalability, and software infrastructure.

1.2.1 The Optimality Gap

The first issue is the *optimality gap*. In heuristic clustering, the numerical value returned by an algorithm may be far from the global optimum, but the user often has no quantitative basis for assessing how far. This problem is especially acute in comparative studies, where algorithmic conclusions can depend on differences in local minima rather than differences in model quality. It also matters in applications where the clustering itself is part of a decision process: a suboptimal partition may distort scientific conclusions,

induce unnecessary operational costs, or mischaracterise heterogeneity in the underlying data.

For K-means, this issue has been examined from several complementary angles. Work on the complexity of the objective establishes that exact optimisation is intrinsically challenging in the worst case (Aloise et al., 2009; Mahajan et al., 2012). Work on algorithmic behaviour shows that iterative schemes can be sensitive to initialisation and data geometry, even when practical performance is often strong (Arthur & Vassilvitskii, 2007; Bottou & Bengio, 1994; Celebi et al., 2013). Related concerns apply to medoid-based algorithms, where the use of representative samples improves robustness and metric generality but does not in itself resolve the optimisation problem (Park & Jun, 2009; Schubert & Rousseeuw, 2021; Velmurugan & Santhanam, 2010). The existence of an optimality gap is therefore not incidental; it is a structural feature of the problem class.

1.2.2 Recent Progress on Exact Large-Scale Clustering

Historically, exact methods based on branch-and-bound (B&B), mixed-integer programming, or decomposition were often viewed as practical only for small benchmark instances. That perception has changed in recent years. For K-centers, Shi et al. (2022) and Ren et al. (2025) report reduced-space spatial B&B algorithms that scale to very large instances and deliver globally optimal or rigorously certified solutions on parallel hardware. For K-means, Hua et al. (2021) build on the reduced-space spatial B&B scheme of Cao and Zavala (2019) and demonstrate globally optimised solutions on datasets with more than 200 000 points using Lagrangian-decomposition B&B. For K-medoids, Ren et al. (2022) combine reduced-space B&B with a tailored Lagrangian relaxation and bound-tightening procedures, and report provably near-optimal solutions on datasets with up to one million samples. These results do not imply that exact clustering is universally easy, but they do show that exact or gap-certified optimisation can now be computationally meaningful far beyond toy examples. Just as importantly, they reveal a second fact that motivates this thesis: recent progress has been achieved through model-specific algorithmic designs that remain dispersed across separate papers, formulations, and software implementations.

1.2.3 Fragmented Methodology Across Models

The second issue is the *model-fragmentation gap*. Existing exact or global methods are commonly developed for one clustering model at a time, one

relaxation strategy at a time, or one algorithmic architecture at a time. As a result, it is difficult to compare K-means, K-centers, and K-medoids within a unified mathematical and computational framework. Differences in notation, feasibility structure, bounding procedures, and implementation assumptions complicate direct methodological comparison.

This fragmentation has practical consequences. When methods are designed in isolation, it becomes difficult to identify which improvements are model-specific and which are structural. It also becomes harder to study how common ingredients—such as decomposition, bound tightening, or distributed search—behave across distinct but related objectives. A unified framework is therefore valuable not only for software engineering reasons, but also for methodological clarity. It makes possible a cleaner comparison of algorithmic components across models that share a common centroid-based interpretation but differ in optimisation structure.

1.2.4 Software and Reproducibility Gap

The third issue is the *software gap*. Julia has become an increasingly important environment for scientific computing because it combines high-level numerical expressiveness with competitive execution speed (Bezanson et al., 2017). This makes it especially attractive for optimisation software that must balance mathematical modelling, performance engineering, and reproducible experimentation. Yet the current Julia clustering ecosystem remains primarily oriented toward heuristic solvers.

Core packages such as `Clustering.jl` (JuliaStats Contributors, 2025) and `MLJ.jl` (Blaom et al., 2020) provide useful abstractions, standard workflows, and integration with the broader machine learning stack, but they do not currently supply a unified interface to globally optimal or gap-certified solvers for K-means, K-centers, and K-medoids. Other Julia packages demonstrate that high-performance clustering-oriented computation is feasible in specialised settings—for example, `GigaSOM.jl` supports large-scale cytometry workflows (Kratochvíl et al., 2020)—yet these tools do not constitute a coherent optimisation framework for the canonical centroid-based models. Thus, the current ecosystem contains strong computational components, but lacks an integrated software layer for exact clustering research.

Taken together, these considerations define the central objective of this thesis: to develop a unified global-optimisation framework for centroid-based

clustering that is mathematically rigorous, computationally scalable, and implemented in a form that supports systematic experimentation and reproducibility. Specifically, this thesis develops a unified Julia-based framework that combines branch-and-bound, problem-specific lower and upper bounds, bound tightening, and distributed-memory parallelism. On the bounding side, the framework draws on decomposition and dual methods related to subgradient optimisation (M. L. Fisher, 1981), the Volume Algorithm (Barahona & Anbil, 2000), and Progressive Hedging (Rockafellar & Wets, 1991).

1.3 Existing Julia Clustering Packages

The current Julia clustering ecosystem comprises a small number of general-purpose packages together with several specialised high-performance tools. From the perspective of this thesis, the key point is not merely that these packages exist, but that they occupy different layers of the computational stack. Some provide clustering algorithms, some provide model-management interfaces, and others provide numerical primitives that support clustering at scale. What remains absent is a unified software layer for globally optimal centroid-based clustering.

1.3.1 Clustering.jl

`Clustering.jl` (JuliaStats Contributors, 2025) is the principal general-purpose clustering package in the Julia ecosystem. It includes implementations of K-means, K-medoids, DBSCAN (Ester et al., 1996), fuzzy C-means, affinity propagation (Frey & Dueck, 2007), hierarchical clustering, and the Markov Clustering Algorithm. The package is designed to interoperate with Julia's distance-computation infrastructure and offers a relatively clean application programming interface for routine clustering analysis. It also includes standard quality measures such as silhouette scores and external partition-comparison metrics, which makes it suitable for applied data analysis and method evaluation.

From the perspective of global optimisation, however, its scope is limited. The package contains heuristic routines only; it does not provide branch-and-bound procedures, mixed-integer formulations, or certified lower and upper bounds for the problems studied in this thesis. Its K-medoids implementation is not designed as a state-of-the-art exact or accelerated medoid solver in the sense of Schubert and Rousseeuw (2021). For K-centers, it

does not expose globally optimal or approximation-guaranteed algorithms as a dedicated solver layer. In short, `Clustering.jl` is valuable as a heuristic baseline and ecosystem anchor, but it does not address the exactness and certification questions that motivate this thesis.

1.3.2 MLJ.jl and the Broader Machine Learning Framework

`MLJ.jl` (Blaom et al., 2020) provides a meta-interface for machine learning models in Julia and plays an important role in standardising model fitting, prediction, evaluation, and pipeline composition. It wraps models from multiple packages under a common interface and supports hyperparameter tuning, validation workflows, and modular pipeline construction. In this sense, `MLJ.jl` addresses interface fragmentation for applied machine learning users.

However, this unification occurs at the level of workflow orchestration rather than global optimisation. The package does not itself provide exact clustering algorithms, global lower and upper bounds, or a common framework for certification across K-means, K-centers, and K-medoids. It is therefore best interpreted as complementary to the aims of this thesis rather than as a substitute for them.

1.3.3 High-Performance and Domain-Specific Packages

Several Julia packages demonstrate the feasibility of large-scale clustering-oriented computation in specialised settings. For example, `GigaSOM.jl` (Kratochvíl et al., 2020) targets high-throughput cytometry applications and illustrates how Julia can support scalable clustering-related workloads in a domain-specific context. Other packages in the Julia ecosystem provide efficient nearest-neighbour search, distance evaluation, and parallel numerical kernels that are directly relevant to the implementation of clustering algorithms.

These tools are important because they show that the ecosystem already contains many of the low-level ingredients needed for high-performance optimisation software. At the same time, they do not by themselves provide a unified mathematical and computational framework for globally solving the canonical centroid-based clustering problems. This distinction motivates the software contribution of the present thesis: rather than adding another heuristic package, the goal is to integrate exact optimisation techniques into a coherent Julia-based framework that can be used for systematic computa-

tional study.

1.4 Contributions and Thesis Organisation

Against this background, the contribution of this thesis is not only to formulate global optimisation models for centroid-based clustering, but also to organise them into a coherent algorithmic and software framework that supports comparison, scalability, and reproducibility. The contributions are fourfold:

- **Unified modelling:** the thesis develops a single optimisation framework that covers K-means, K-centers, and K-medoids with consistent notation, mathematical formulations, and evaluation criteria.
- **Algorithmic design:** it introduces model-aware bounding and search mechanisms, including closed-form bounds, scenario grouping, dual-enhanced decompositions, feasibility-based tightening, and branch-and-bound procedures coupled with Lagrangian decomposition.
- **Software and reproducibility:** it implements these methods in a distributed-memory Julia environment with structured logging, command-line tools, batch experimentation support, and reproducible computational outputs.
- **Computational evidence:** it provides experiments designed to report global optimality certificates or tight optimality gaps and to quantify the discrepancy between heuristic solutions and certified solutions across multiple datasets.

The remainder of this thesis is organised as follows. Chapter 2 introduces unified formulations and a two-stage perspective for K-means, K-centers, and K-medoids. Chapter 3 presents the reduced-space branch-and-bound framework together with lower bounds, upper bounds, and acceleration mechanisms. Chapter 4 describes the software architecture and computational study design, with particular emphasis on scalability and reproducibility. Chapter 5 presents the computational experiments and reporting conventions used to evaluate certification quality, method trade-offs, and parallel performance.

Chapter 2

Problem Formulations and Mathematical Background

Consider a dataset

$$X = \{x_s\}_{s=1}^{n} \subset \mathbb{R}^p$$

consisting of n samples, which we aim to partition into K clusters indexed by $\mathcal{K} = \{1, 2, \ldots, K\}$. Each cluster is represented by a center μ_k, where in K-means $\mu_k \in \mathbb{R}^p$ can lie anywhere in space, while in K-centers and K-medoids each μ_k must be chosen from the dataset X. All three methods optimize an objective of the form

$$\min_{\{\mu_k\}_{k\in\mathcal{K}}} \mathcal{F}\Big(\{\min_{k\in\mathcal{K}} d(x_s, \mu_k)\}_{s=1}^{n}\Big),$$

where for K-means and K-centers $d(x, \mu) = \|x - \mu\|_2^2$. For K-medoids, the mixed-integer model in this chapter can be written with a precomputed dissimilarity matrix, but the reduced-space framework developed in Chapters 3–5 specializes to squared Euclidean distances on vector-valued data. Concretely,

$$\mathcal{F}_{\text{kmeans}} = \sum_{s=1}^{n} \min_{k\in\mathcal{K}} \|x_s - \mu_k\|_2^2, \quad \mu_k \in \mathbb{R}^p,$$

$$\mathcal{F}_{\text{kcenters}} = \max_{s=1}^{n} \min_{k\in\mathcal{K}} \|x_s - \mu_k\|_2^2, \quad \mu_k \in X,$$

$$\mathcal{F}_{\text{medoids}} = \sum_{s=1}^{n} \min_{k\in\mathcal{K}} \|x_s - \mu_k\|_2^2, \quad \mu_k \in X.$$

2.1 Scope and Relation to the Source Papers

This thesis is best understood as a unification and reproduction study of three recent exact-clustering research threads rather than as the introduction of three unrelated new algorithms. The K-means solver architecture

follows the reduced-space branch-and-bound and Lagrangian-decomposition approach of Hua et al. (2021). The K-centers track draws primarily on the reduced-space spatial branch-and-bound method and feasibility-based bound-tightening ideas developed by Shi et al. (2022), with the large-scale computational perspective further extended by Ren et al. (2025). The K-medoids track follows the reduced-space branch-and-bound framework of Ren et al. (2022), which combines branching on medoid regions with a tailored Lagrangian relaxation and probing-based tightening.

The contribution of the thesis is therefore methodological in a different sense. It places these model-specific exact methods into one mathematical vocabulary, one software architecture, and one experimental protocol. This unification matters because the three models share the same high-level search template—branch on the first-stage center variables, compute certified lower bounds, generate feasible upper bounds, tighten the search region, and parallelize expensive bound computations—while differing in the structure of their reduced costs and feasibility restrictions. Writing them side by side makes clear which ideas are generic and which depend on the objective or on the "centers-on-samples" constraint.

The relationship between the three source papers and the unified framework developed in this thesis can be summarized directly in prose. For K-means, the framework follows Hua et al. (2021): it branches on continuous center boxes, uses an analytic lower bound strengthened by scenario grouping and Lagrangian decomposition, and accelerates computation through parallel subproblems and model-aware upper bounds. For K-centers, the framework draws on Shi et al. (2022), with the large-scale extension of Ren et al. (2025): it treats centers as samples selected through center boxes, uses a closed-form decomposable lower bound for the max-type objective, and relies on feasibility-based bound tightening together with reduced-space branching. For K-medoids, the framework follows Ren et al. (2022): it branches on medoid regions with sample-restricted centers, uses a tailored Lagrangian relaxation whose dual subproblem can be evaluated analytically, and accelerates the search through probing and feasibility-based tightening on candidate medoids.

From the standpoint of the remaining chapters, this summary serves two purposes. First, it clarifies scope: the unified framework does not claim that the same lower bound or the same tightening rule applies unchanged to all three models. Second, it clarifies contribution: the thesis adds value by standardizing notation, isolating reusable software interfaces, and enabling

a controlled computational comparison across methods that were originally reported separately.

2.2 Mixed-Integer Optimization Formulations

To derive globally optimized solutions, we formulate the three clustering problems as mixed-integer optimization models. Let $\mathcal{S} = \{1, 2, \ldots, n\}$ be the sample index set. Let $z_{s,k} \in \{0, 1\}$ denote assignment variables such that $z_{s,k} = 1$ if sample x_s belongs to cluster k, and 0 otherwise.

The K-means problem minimizes the within-cluster sum of squared errors. A mixed-integer second-order cone programming (MISOCP) formulation is:

$$\min_{\mu,d,z} \quad \sum_{s\in\mathcal{S}} d_{s,*} \tag{2.1}$$

$$\text{s.t.} \quad -N(1 - z_{s,k}) \le d_{s,*} - d_{s,k} \le N(1 - z_{s,k}) \tag{2.2}$$

$$d_{s,k} \ge \|x_s - \mu_k\|_2^2 \tag{2.3}$$

$$\sum_{k\in\mathcal{K}} z_{s,k} = 1 \tag{2.4}$$

$$z_{s,k} \in \{0, 1\} \tag{2.5}$$

$$s \in \mathcal{S},\ k \in \mathcal{K}. \tag{2.6}$$

Here, $d_{s,k}$ represents the distance from sample s to center k, $d_{s,*}$ denotes the distance from sample s to its assigned center, and N is a sufficiently large constant.

The K-centers problem minimizes the maximum within-cluster squared distance. A min-max MINLP formulation, written in notation consistent with

the implementation-facing discussion in later chapters, is:

$$\min_{\mu,d,z,\lambda,d_*} \quad d_* \tag{2.7}$$

$$\text{s.t.} \quad d_{s,k} \geq \|x_s - \mu_k\|_2^2 \tag{2.8}$$

$$-N_1(1 - z_{s,k}) \leq d_{s,*} - d_{s,k} \leq 0 \tag{2.9}$$

$$d_* \geq d_{s,*} \tag{2.10}$$

$$\sum_{k \in \mathcal{K}} z_{s,k} = 1 \tag{2.11}$$

$$z_{s,k} \in \{0, 1\} \tag{2.12}$$

$$-N_2(1 - \lambda_s^k) \leq x_s - \mu_k \leq N_2(1 - \lambda_s^k) \tag{2.13}$$

$$\sum_{s \in \mathcal{S}} \lambda_s^k = 1 \tag{2.14}$$

$$\lambda_s^k \in \{0, 1\} \tag{2.15}$$

$$z_{s,k} \geq \lambda_s^k \tag{2.16}$$

$$s \in \mathcal{S},\ k \in \mathcal{K}. \tag{2.17}$$

Here, $d_{s,k}$ is the squared distance from sample x_s to center μ_k, $d_{s,*}$ is the squared distance from sample x_s to the center of its assigned cluster, and d_* is the maximum assigned squared distance across all samples. Constraint (2.9) ensures that $d_{s,*} = d_{s,k}$ when $z_{s,k} = 1$ and $d_{s,*} \leq d_{s,k}$ otherwise; Constraint (2.10) then makes d_* an epigraph variable for the maximum assigned distance. Constraint (2.11) guarantees that each sample is assigned to exactly one cluster. Constraints (2.13)–(2.16) encode the "centers on samples" requirement: if $\lambda_s^k = 1$, then $\mu_k = x_s$; each center selects exactly one sample; and a selected center is assigned to its own cluster.

In K-medoids clustering, centers (medoids) must be selected directly from X. Using pairwise squared Euclidean distances $d_{s,j} = \|x_s - x_j\|_2^2$ and medoid

indicators $y_j \in \{0,1\}$, the MILP formulation is:

$$\min_{z,y} \quad \sum_{s\in\mathcal{S}}\sum_{j\in\mathcal{S}} d_{s,j} z_{s,j} \tag{2.18}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{S}} z_{s,j} = 1, \quad \forall s \in \mathcal{S} \tag{2.19}$$

$$\sum_{j\in\mathcal{S}} y_j = K \tag{2.20}$$

$$z_{s,j} \le y_j, \quad \forall s, j \in \mathcal{S} \tag{2.21}$$

$$z_{s,j}, y_j \in \{0,1\}, \quad \forall s, j \in \mathcal{S}. \tag{2.22}$$

2.3 Two-Stage Optimization Structure

All three models admit a two-stage interpretation once centers are fixed. Throughout the mathematical development, we index centers as a $K \times p$ array $\mu = \{\mu_k\}_{k\in\mathcal{K}}$, with the additional feasibility restriction $\mu_k \in X$ for K-centers and K-medoids.

For the sum-type objectives (K-means and K-medoids), the reduced problem is

$$v^* = \min_{\mu\in\mathcal{M}} \sum_{s\in\mathcal{S}} Q_s(\mu), \tag{2.23}$$

where

$$Q_s(\mu) = \min_{k\in\mathcal{K}} c_s(x_s, \mu_k), \qquad c_s(x_s,\mu_k) = \begin{cases} \|x_s - \mu_k\|_2^2 & \text{(K-means)}, \\ \|x_s - \mu_k\|_2^2 & \text{(K-medoids)}. \end{cases} \tag{2.24}$$

For K-centers, the outer aggregation is max-type and is written in epigraph form as

$$v^*_{\mathrm{KC}} = \min_{\mu\in X^K,\, r} \; r \quad \text{s.t.} \quad Q_s^{\mathrm{KC}}(\mu) \le r, \; \forall s \in \mathcal{S}, \tag{2.25}$$

with sample-level reduced cost

$$Q_s^{\mathrm{KC}}(\mu) = \min_{k\in\mathcal{K}} \|x_s - \mu_k\|_2^2. \tag{2.26}$$

Chapter 3

Algorithmic Framework

This chapter presents the reduced-space branch-and-bound framework used to compute globally optimized clustering solutions. The method combines closed-form bounds, decomposition-based tightenings, and node-level acceleration.

3.1 Branch-and-Bound Framework

Algorithm 1 maintains a collection of rectangular regions in the center space and iteratively refines the most promising node. Let $\beta(\cdot)$ and $\alpha(\cdot)$ denote lower and upper bound functions. Under standard consistency and shrinkage assumptions, the global lower bound is nondecreasing, the global upper bound is nonincreasing, and convergence is achieved when the gap is below tolerance.

3.2 Lower Bounding Procedures

3.2.1 Two-Stage Decomposition

All three clustering variants admit a two-stage structure once centers are fixed, but the outer aggregation depends on the model. For a node region $M \subseteq M_0$:

$$z(M) = \min_{\mu \in M \cap \mathcal{C}} \sum_{s \in \mathcal{S}} Q_s(\mu), \qquad \text{(KM, MD)} \tag{3.1}$$

and for K-centers,

$$z^{\mathrm{KC}}(M) = \min_{\mu \in M \cap \mathcal{C},\, r} \; r \quad \text{s.t.} \quad Q_s(\mu) \le r, \; \forall s \in \mathcal{S}. \tag{3.2}$$

Algorithm 1 RS-BB Framework for Clustering

Initialize: node set $\mathcal{N} = \{M_0\}$, incumbent $UB = \infty$, tolerance $\epsilon > 0$
Compute initial bounds: $LB_0 = \beta(M_0)$, $LB = LB_0$, $UB = \min\{UB, \alpha(M_0)\}$
while $\mathcal{N} \neq \emptyset$ and $UB - LB > \epsilon$ **do**
  Select $\hat{M} \in \mathcal{N}$ with minimum lower bound
  Remove $\hat{M}$ from $\mathcal{N}$
  Compute local bounds: $LB_{\text{local}} = \beta(\hat{M})$, $UB_{\text{local}} = \alpha(\hat{M})$
  Update global upper bound: $UB = \min\{UB, UB_{\text{local}}\}$
  if $LB_{\text{local}} \leq UB - \epsilon$ **then**
    Branch on variable $\mu_{k,a}$ with largest interval width
    Partition $\hat{M}$ into children M_1, M_2
    Add feasible children to $\mathcal{N}$
  end if
  Prune all $M' \in \mathcal{N}$ with $\beta(M') > UB$
  if $\mathcal{N} = \emptyset$ **then**
    break
  else
    Set $LB = \min_{M' \in \mathcal{N}} \beta(M')$
  end if
end while

Here $\mathcal{C}$ denotes problem-specific center constraints. Introducing sample-specific center copies μ_s yields a lifted model for the sum-type objectives:

$$z(M) = \min_{\{\mu_s\}_{s\in\mathcal{S}}} \quad \sum_{s\in\mathcal{S}} Q_s(\mu_s) \tag{3.3}$$

$$\text{s.t.} \quad \mu_s = \mu_{s+1}, \quad s \in \{1,\dots,|\mathcal{S}|-1\}, \tag{3.4}$$

$$\mu_s \in M \cap \mathcal{C}. \tag{3.5}$$

For K-centers, the same lifting is applied inside the epigraph constraints by replacing $Q_s(\mu) \le r$ with $Q_s(\mu_s) \le r$ for each sample s. Relaxing the non-anticipativity constraints then leads to decomposable lower bounds.

3.2.2 Closed-Form and Decomposition Bounds

Let $\mathcal{M} = \prod_{k\in\mathcal{K}} \mathcal{M}^k$ with $\mathcal{M}^k = \prod_{a=1}^{p} [\mu^l_{k,a}, \mu^u_{k,a}]$. Throughout the reduced-space framework in this chapter, all three models are treated on vector-valued data in $\mathbb{R}^p$ using squared Euclidean distances. In particular, the K-medoids results here specialize the more general dissimilarity-matrix MILP from Chapter 2 to the squared Euclidean setting so that coordinate-wise box relaxations and midpoint projections remain well defined. Define

$$\mathrm{mid}\{L,u,U\} := \min\{\max\{u,L\},U\}.$$

Using the sum-type and max-type formulations introduced in Chapter 2, a decomposable basic bound is

$$\beta^{\text{basic}}(\mathcal{M}) = \begin{cases} \displaystyle\sum_{s\in\mathcal{S}} \min_{\mu_s\in\mathcal{M}} Q_s(\mu_s), & \text{sum-type objectives (KM, MD),} \\ \displaystyle\max_{s\in\mathcal{S}} \min_{\mu_s\in\mathcal{M}} Q_s(\mu_s), & \text{max-type objective (KC).} \end{cases} \tag{3.6}$$

The projected distance term has closed form:

$$d^{\min}_{ks}(\mathcal{M}) := \min_{\mu_k\in\mathcal{M}^k} \|x_s - \mu_k\|_2^2 = \sum_{a=1}^{p} \Big(x_{s,a} - \mathrm{mid}\{\mu^l_{k,a}, x_{s,a}, \mu^u_{k,a}\}\Big)^2. \tag{3.7}$$

KM: closed-form and Lagrangian bounds

For K-means,

$$\beta^{\text{KM}}_{\text{basic}}(\mathcal{M}) = \sum_{s\in\mathcal{S}} \min_{k\in\mathcal{K}} d^{\min}_{ks}(\mathcal{M}). \tag{3.8}$$

Dualizing non-anticipativity yields

$$\beta^{\mathrm{KM}}_{\mathrm{LD}}(\mathcal{M}) = \max_{\lambda} \sum_{s\in\mathcal{S}} \beta^{\mathrm{KM}}_{s}(\mathcal{M},\lambda), \qquad \beta^{\mathrm{KM}}_{\mathrm{basic}}(\mathcal{M}) \le \beta^{\mathrm{KM}}_{\mathrm{LD}}(\mathcal{M}) \le z(\mathcal{M}). \tag{3.9}$$

We solve the dual using standard nonsmooth methods including the subgradient method (M. L. Fisher, 1981), the Volume Algorithm (Barahona & Anbil, 2000), and Progressive Hedging (Rockafellar & Wets, 1991).

Scenario grouping further yields

$$\beta^{\mathrm{KM}}_{\mathrm{basic}}(\mathcal{M}) \le \beta^{\mathrm{KM}}_{\mathrm{SG}}(\mathcal{M}) \le \beta^{\mathrm{KM}}_{\mathrm{SG+LD}}(\mathcal{M}) \le z(\mathcal{M}).$$

KC: closed-form bound

For K-centers,

$$\beta^{\mathrm{KC}}_{\mathrm{basic}}(\mathcal{M}) = \max_{s\in\mathcal{S}} \min_{k\in\mathcal{K}} d^{\min}_{ks}(\mathcal{M}). \tag{3.10}$$

MD: continuous bound and combinatorial LD bound

Relaxing medoid discreteness gives

$$\beta^{\mathrm{MD}}_{\mathrm{basic}}(\mathcal{M}) = \sum_{s\in\mathcal{S}} \min_{k\in\mathcal{K}} d^{\min}_{ks}(\mathcal{M}). \tag{3.11}$$

Dualizing assignment constraints in (2.18)–(2.22) leads to

$$\beta^{\mathrm{MD}}_{\mathrm{LD}}(\mathcal{M}) = \max_{\lambda} \beta^{\mathrm{MD}}_{\mathrm{LD}}(\mathcal{M},\lambda), \qquad \beta^{\mathrm{MD}}_{\mathrm{basic}}(\mathcal{M}) \le \beta^{\mathrm{MD}}_{\mathrm{LD}}(\mathcal{M}) \le z(\mathcal{M}). \tag{3.12}$$

3.3 Upper Bounds

Upper bounds are obtained from feasible center sets and objective evaluation.

3.3.1 KM: basic and local-refined upper bounds

For a feasible candidate $\hat{\mu} \in \mathcal{M}$:

$$\alpha^{\mathrm{KM}}_{\mathrm{basic}}(\mathcal{M};\hat{\mu}) = \sum_{s\in\mathcal{S}} \min_{k\in\mathcal{K}} \|x_s - \hat{\mu}_k\|_2^2. \tag{3.13}$$

A local refinement solves:

$$\begin{aligned} \min_{\mu \in \mathcal{M},\, b} \quad & \sum_{s \in \mathcal{S}} \sum_{k \in \mathcal{K}} b_{s,k} \, \|x_s - \mu_k\|_2^2 \\ \text{s.t.} \quad & \sum_{k \in \mathcal{K}} b_{s,k} = 1, \quad 0 \le b_{s,k} \le 1, \quad \forall s \in \mathcal{S},\, k \in \mathcal{K}. \end{aligned} \tag{3.14}$$

The refined upper bound is:

$$\alpha_{\text{loc}}^{\text{KM}}(\mathcal{M}) = \sum_{s \in \mathcal{S}} \min_{k \in \mathcal{K}} \|x_s - \mu_k^{\text{loc}}\|_2^2. \tag{3.15}$$

3.3.2 KC and MD upper bounds

For KC, we use farthest-first traversal (FFT) candidates and box-midpoint-guided snapping:

$$\alpha_{\text{FFT}}^{\text{KC}}(\mathcal{M}) = \max_{s \in \mathcal{S}} \min_{k \in \mathcal{K}} \|x_s - \hat{\mu}_k^{\text{FFT}}\|_2^2, \tag{3.16}$$

$$\alpha_{\text{mid}}^{\text{KC}}(\mathcal{M}) = \max_{s \in \mathcal{S}} \min_{k \in \mathcal{K}} \|x_s - \hat{\mu}_k^{\text{mid}}\|_2^2. \tag{3.17}$$

For MD, we use root-node initialization and child-node snapping refinements:

$$\alpha_{\text{root}}^{\text{MD}}(\mathcal{M}) = \sum_{s \in \mathcal{S}} \min_{k \in \mathcal{K}} \|x_s - \tilde{\mu}_k\|_2^2, \tag{3.18}$$

$$\alpha_{\text{child}}^{\text{MD}}(\mathcal{M}) = \sum_{s \in \mathcal{S}} \min_{k \in \mathcal{K}} \|x_s - \tilde{\mu}_k^{\text{snap}}\|_2^2. \tag{3.19}$$

3.4 Acceleration and Branching

3.4.1 Optimization-Based Bound Tightening

At the root box $\mathcal{M}_0$, optimization-based bound tightening (OBBT) solves auxiliary minimization/maximization problems for each coordinate $\mu_{k,a}$ under the current relaxation and incumbent cut. The resulting interval updates are inherited by descendants.

3.4.2 Probing

Given a tentative interval restriction $I \subseteq [\mu^l_{k,a}, \mu^u_{k,a}]$, we recompute a fast bound

$$\widehat{\mathrm{LB}}(\mathcal{M} \restriction I) = \sum_s \min_{k'} d^{\min}_{k's}(\mathcal{M} \restriction I).$$

If $\widehat{\mathrm{LB}} \geq \mathrm{UB}$, the interval is permanently shrunk; otherwise the probe is discarded.

3.4.3 Branching Rule

Branching selects the coordinate with maximal spread,

$$(k^\star, a^\star) \in \arg\max_{k,a} (\mu^u_{k,a} - \mu^l_{k,a}),$$

and performs a midpoint split into left/right child boxes.

Chapter 4

Software Design and Computational Study

This chapter describes how the formulations of Chapter 2 and the search procedures of Chapter 3 are translated into software. The implementation is written in Julia and organized as the package `GO_Clustering.jl`. The objective is not merely to provide an executable version of the proposed methods, but to preserve the logical separation between lower and upper bounding, domain reduction, branching, parallel communication, and experimental workflow management in a form that is extensible and reproducible.

4.1 Software Overview and Design Goals

The Julia package `GO_Clustering.jl` is designed to solve three centroid-based clustering problems—K-means, K-medoids, and K-centers—with certified global optimality or rigorously controlled optimality gaps. In contrast to heuristic procedures such as Lloyd's algorithm or Partitioning Around Medoids (PAM), which return only locally optimal or approximation- quality solutions, the implementation couples a branch-and-bound (B&B) engine with provably valid lower bounds, heuristic but feasible upper bounds, and domain-reduction procedures. The resulting search is executed in parallel through the Message Passing Interface (MPI).

Equally important, the package should be interpreted as a *reproduction and integration framework* for the exact methods studied in this thesis. Its role is not simply to provide code for isolated algorithms, but to re-implement the K-means, K-centers, and K-medoids methods drawn from the source papers in a shared software environment. This common environment makes it possible to compare bound quality, branching behavior, tightening effectiveness, and parallel scalability under consistent data handling, logging, and stopping rules. In that sense, the software contribution is part of the

research contribution: it turns three related but separately reported algorithmic designs into one reproducible computational laboratory.

The software architecture is guided by three design goals.

G1: Certified global optimality. Every component of the search tree is designed so that the final lower bound is mathematically valid for the original nonconvex clustering problem. This requirement rules out expedient shortcuts that are acceptable in heuristic packages but incompatible with exact optimization. It also motivates a strict separation between certificate-producing routines, such as lower bounds and bound tightening, and quality-improving routines, such as heuristic upper bounds.

G2: Algorithmic extensibility. Although the three clustering models share a common centroid-based structure, their lower bounds, domain reductions, and branching rules differ materially. Rather than build three monolithic solvers, the package factors the search into four interchangeable layers for lower bounding, upper bounding, bound tightening, and branching. These layers are selected at runtime through the parameters `algo` and `method`. This makes the addition of a new clustering variant or a new lower bound a localized software change rather than a complete rewrite of the search engine.

G3: Parallel scalability with deterministic semantics. The implementation uses a Single-Program-Multiple-Data (SPMD) model on top of `MPI.jl`. Each rank holds the same active search tree and the same incumbent upper bound, so that pruning decisions are logically identical across processes. This choice avoids the communication bottleneck of a pure master–worker architecture while maintaining deterministic behavior once the random seed and process count are fixed.

The remainder of the chapter is organized as follows. Section 4.2 introduces the software architecture and module graph. Section 4.3 describes the core node data structures. Section 4.4 presents the B&B orchestrator. Section 4.5 summarizes the pluggable algorithmic components. Section 4.6 discusses the distributed parallel layer and an implementation lesson related to correctness. Section 4.7 explains the experimental tooling, reproducibility infrastructure, and extension workflow. Section 4.8 concludes with software-availability remarks.

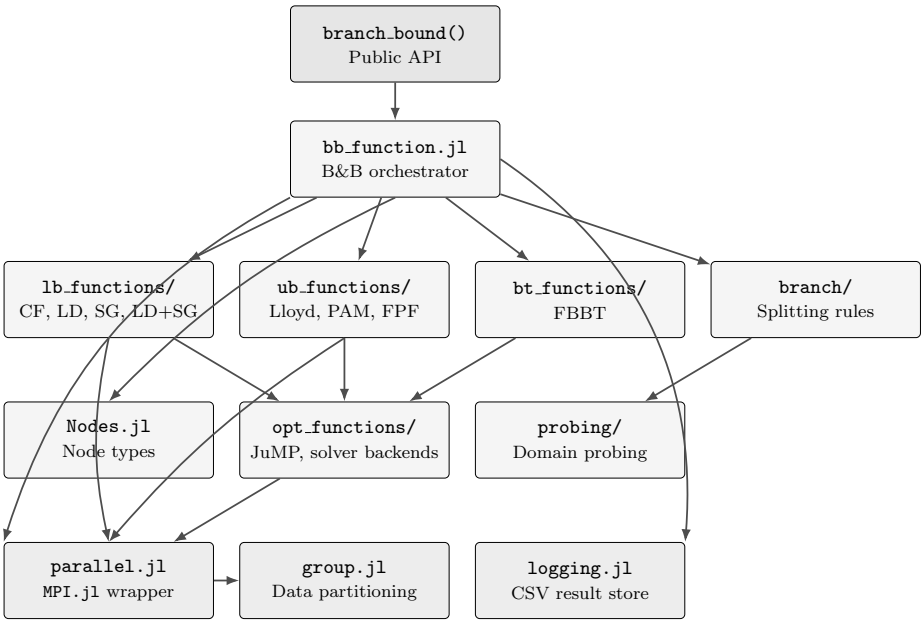

Figure 4.1: Module dependency graph. Arrows point from caller to callee. The layers correspond to the public interface, the B&B orchestration layer, the pluggable algorithmic components, the numerical solver abstraction, and the supporting infrastructure.

4.2 System Architecture

The package follows a layered architecture whose dependencies flow in one direction only; see Figure 4.1. The public interface exposes the entry point `branch_bound`. An orchestration layer implements the search loop and node management. Below it, an algorithmic component layer provides problem-specific lower bounds, upper bounds, bound tightening, and branching rules. A numerical layer encapsulates the JuMP formulations that interact with external solver backends. Finally, an infrastructure layer provides MPI wrappers, data partitioning utilities, and structured logging. Table 4.1 summarizes the responsibilities of the principal software modules.

External dependencies. The implementation builds on the Julia optimization ecosystem rather than re-implementing standard primitives. Mathematical models are expressed through `JuMP.jl`; mixed-integer subproblems are solved by IBM ILOG CPLEX Optimization Studio (International Busi-

ness Machines Corporation, 2022), while optional continuous refinements are handled through separate nonlinear optimization routines when enabled. Heuristic warm starts use `Clustering.jl` for Lloyd-style K-means and `Metaheuristics.jl` for evolutionary medoid search. Distance computations rely on `Distances.jl`, while `StaticArrays.jl` and `SparseArrays` reduce memory traffic in the innermost routines. The parallel layer is implemented through `MPI.jl`, which links to a system MPI installation.

4.3 Core Data Structures

A B&B node must store enough information to compute local bounds, generate children, and be serialized across MPI ranks. Three design considerations shape the node representation.

First, each node stores a box-shaped relaxation of the cluster centers as a pair of $p \times K$ matrices, `lower` and `upper`. The mathematical development in Chapters 2 and 3 indexes centers as a $K \times p$ array μ, but the implementation stores the transpose for locality and column-wise center access: column k corresponds to center μ_k and row a corresponds to coordinate a. For K-means and K-centers, these matrices describe admissible coordinate ranges for the continuous center variables. For K-medoids, where centers must coincide with sample points, the same box acts as an over-approximation that restricts which data indices may remain feasible medoids.

Second, the three clustering problems are structurally different enough that a single universal node object would either waste memory or force the inclusion of fields that are irrelevant for two of the three cases. The package therefore introduces an abstract supertype `Node` together with three concrete subtypes declared in `src/Nodes.jl`; see Listing 4.1.

```
abstract type Node end

mutable struct Node_kmeans <: Node
    lower::Union{Nothing,Matrix{Float64}}
    upper::Union{Nothing,Matrix{Float64}}
    level::Int
    LB::Float64
    groups
    lambda
    group_centers
end

mutable struct Node_kcenter <: Node
```

```
    lower::Union{Nothing,Matrix{Float64}}
    upper::Union{Nothing,Matrix{Float64}}
    level::Int
    LB::Float64
    assign
    center_cand
end

mutable struct Node_kmedoids <: Node
    lower::Union{Nothing,Matrix{Float64}}
    upper::Union{Nothing,Matrix{Float64}}
    level::Int
    LB::Float64
    lambda::Union{Nothing,Vector{Float64}}
    group_centers::Union{Nothing,Vector{Int}}
    fixed_centers::Union{Nothing,Vector{Int}}
    assign_idx::Union{Nothing,SparseVector{Int64,Int64}}
    best_lambda::Union{Nothing,Vector{Float64}}
    best_alpha::Union{Nothing,Vector{Float64}}
    best_alpha_index_k::Union{Nothing,Vector{Int}}
    fbbt_type::Bool
    bVarIds::Union{Nothing,Vector{Int}}
    CF_LB::Float64
end
```

Listing 4.1: Node type hierarchy used by the software implementation.

Third, nodes are immutable wherever the orchestrator broadcasts them. An immutable `struct` can be serialized and reconstructed without violating internal invariants. The only exception is `Node_kcenter`, which is kept mutable because its assignment and center-candidate fields are repeatedly refined during feasibility-based bound tightening; reallocating the entire object at each refinement step was measurably slower in profiling.

The active search tree is stored as a plain vector of nodes on each rank. The routine `getGlobalLowerBound(nodeList)` scans this vector and returns the node with the smallest lower bound, thereby implementing a best-bound-first strategy. For the problem sizes targeted in this thesis, node bounding dominates node-selection cost, so more elaborate priority-queue data structures were unnecessary.

4.4 The Branch-and-Bound Orchestrator

The orchestrator is the routine `branch_bound` implemented in `src/bb_function.jl`. It is the only function that directly manipulates the global search

Algorithm 2 Main B&B routine `branch_bound(X, k, algo, method, mode, solver)`

Initialize MPI and create the communicator.
Optionally normalize X and compute the initial box (L_0, U_0).
Build the root node from `algo`, compute an initial heuristic UB, and set `nodeList`$\leftarrow$[root].
Broadcast problem dimensions and root-node data to all ranks.
while `nodeList` $\neq \emptyset$, time remains, and $iter <$ `maxiter` **do**
  On the root rank, call `getGlobalLowerBound(nodeList)` and broadcast the selected node index.
  Remove the selected node ν from `nodeList`.
  if $(UB - LB_{\text{node}})/\min\{|LB_{\text{node}}|, |UB|\} \leq$ `mingap` **then**
    Record convergence information and continue.
  end if
  Compute the local lower bound $LB_{\text{node}} \leftarrow \text{LB}_{\texttt{algo,method}}(X, K, \nu, UB)$.
  Compute an upper bound candidate $UB_{\text{node}} \leftarrow \text{UB}_{\texttt{algo}}(X, \nu)$.
  if $UB_{\text{node}} < UB$ **then**
    Set $UB \leftarrow UB_{\text{node}}$ and prune nodes whose lower bounds exceed the incumbent.
  end if
  Select the widest box coordinate and branch at its midpoint.
  Broadcast the updated `nodeList` and UB to all ranks.
  Set $iter \leftarrow iter + 1$.
end while
Undo any normalization and return centers, the final UB, and the convergence trace.

state; all lower-bound, upper-bound, tightening, and branching routines operate on an individual node provided by the orchestrator. Algorithm 2 summarizes the resulting control flow.

Two design choices deserve emphasis.

Dispatch through explicit keys. The orchestrator uses the strings `algo` and `method` to decide which lower bound, upper bound, tightening routine, and branching rule to activate. Julia's multiple dispatch could instead specialize directly on the node type, but explicit dispatch inside the orchestrator makes the global search logic easier to read and debug. For an algorithmic

research codebase, this transparency is often more valuable than maximal abstraction.

Best-bound-first with synchronized state. Node selection, incumbent updates, and branching are executed on the root rank and then broadcast to all other ranks. As a result, every process starts the next iteration with byte-identical search state. This guarantees consistent pruning and tie breaking, while still allowing the expensive lower-bound computations to be parallelized.

4.5 Algorithmic Components

The component layer mirrors the mathematical decomposition developed in Chapter 3. Rather than treating individual helper functions as separate scientific contributions, the software groups routines according to the four roles that recur in Algorithm 1: lower bounding, upper bounding, domain reduction, and branching. This organization also follows the three source-paper tracks summarized in Chapter 2. In the thesis context, what matters is the mapping from mathematical objects to software modules, not a catalogue of helper-function names.

4.5.1 Lower bounds

Table 4.2 summarizes the lower-bound methods exposed through the `method` keyword. Each routine takes a node together with the current incumbent UB, and returns a valid lower bound plus any auxiliary state that should be inherited by the node's children.

Mathematical correspondence and software responsibility. The lower-bound layer, implemented mainly in `src/lb_functions/`, realizes the decompositions of Section 3.2.2. Across all three models, the common primitive is the projected-distance expression (3.7). The K-means module aggregates these terms according to the sum-type bound (3.8), and its stronger grouped/Lagrangian variants implement the dual and scenario-grouping ideas summarized in (3.9). Computationally, this layer is responsible not only for producing a bound value, but also for carrying forward reusable state such as group partitions and multiplier vectors so that child nodes do not rebuild the dual state from scratch.

K-centers reuse the same projected-distance primitive but switch to the max-type aggregation of (3.10), reflecting the epigraph formulation introduced in Chapter 2. This is why the K-centers code path differs structurally from the K-means and K-medoids paths even when many geometric calculations are shared. For K-medoids, the lower-bound layer combines the continuous box relaxation (3.11) with the combinatorial Lagrangian strengthening (3.12). Here the implementation must also track feasible medoid candidate sets induced by the discrete assignment model (2.18)–(2.22). In software terms, the lower-bound layer is therefore both a numerical- evaluation layer and a state-propagation layer.

4.5.2 Upper bounds

The upper-bound layer, implemented in `src/ub_functions/`, is organized around feasible incumbent generation rather than exact local optimality. Its role is to instantiate the model-specific constructions from the upper-bound part of Chapter 3: K-means candidates are evaluated and optionally refined according to (3.13)–(3.15), while K-centers and K-medoids use the feasible candidate rules summarized in Section 3.3.2 and equations (3.16)–(3.19). From a software-design perspective, all upper-bound modules share the same contract: they return candidate centers together with a verified objective value, while the orchestrator alone decides whether the incumbent should be replaced. This separation keeps heuristic randomness local and preserves the certification logic of the main B&B loop.

4.5.3 Bound tightening and probing

The domain-reduction layer, implemented through `src/bt_functions/` and `src/probing/`, sits between lower bounding and branching. It uses the incumbent upper bound to rule out parts of the current node that cannot contain an improving solution. For continuous-center models, this means shrinking coordinate boxes while preserving compatibility with the lower-bound formulas of Section 3.2.2, especially the projected-distance expression (3.7) and the model-specific aggregations in (3.8) and (3.10). For K-medoids, domain reduction acts on kernel regions rather than purely continuous intervals, so the tightening logic must remain consistent with the discrete feasibility structure of (2.19)–(2.22) and with the dual certificates behind (3.12). Probing extends this idea by explicitly testing prospective sub-boxes before branching. Computationally, these modules are valuable not because they introduce a new objective, but because they reduce tree

size and improve the quality of inherited node state.

4.5.4 Branching

The branching layer is intentionally simple. It implements the partition step in Algorithm 1 by selecting a wide coordinate range and splitting it, while delegating all model-specific consequences to the node- update logic. For K-means and K-centers, branching mainly creates child boxes and carries forward symmetry restrictions; for K-medoids, branching must additionally recompute feasible medoid candidate sets under the new geometry. The design choice here is deliberate: the thesis does not claim a novel branching rule, but rather shows how a small, transparent branching mechanism can be combined with stronger lower bounds and tightening modules to obtain an effective global search.

4.5.5 Solver abstraction

Finally, the numerical subproblems are isolated in `src/opt_functions/`. This layer provides the JuMP/solver realizations of the mathematical models introduced earlier in the thesis. It includes the local K-means refinement model corresponding to (3.14), the grouped optimization subproblems needed by the decomposition bounds of Section 3.2.2, and the projection, snapping, and objective-evaluation subproblems used when constructing feasible incumbents. Structurally, this separation is important: the orchestrator and component layers can be written in terms of nodes, bounds, and certificates, while solver-specific code remains localized. That keeps the software architecture aligned with the mathematical architecture of Chapters 2 and 3, and it makes later extensions or solver replacement a local change rather than a whole-codebase rewrite.

4.6 Distributed Parallel Layer

The infrastructure layer distinguishes `GO_Clustering.jl` from a single-node clustering implementation. The target computing environment is an HPC cluster with many processes, so the communication model must preserve both scalability and correctness.

4.6.1 SPMD rather than master–worker

Julia's standard `Distributed` library uses a master–worker pattern in which the master dispatches tasks and collects results. That model is convenient for small jobs, but it creates a communication bottleneck at scale. The present implementation instead uses MPI in SPMD mode: every rank runs the same program with the same arguments and maintains a synchronized copy of the search state. Ranks differ only through the communicator identity returned by `parallel.myid()`. Collective operations handle reductions and state propagation.

4.6.2 Custom collective operations

The file `src/parallel.jl` wraps `MPI.jl` and adds several operations needed by the B&B engine for arbitrary Julia objects.

- `bcast(x)` is a wrapper over `MPI.bcast` that handles `Nothing` on non-root ranks. It is used to synchronize the active node list, the incumbent upper bound, multiplier updates, and convergence flags.
- `allcollect(x)` and `collect(x)` gather Julia objects from all ranks by serializing each contribution to a byte buffer, exchanging buffer sizes, and then exchanging payloads through collective communication.
- `spread(x,list_all)` performs the reverse operation: it partitions a vector held on the root rank and distributes the corresponding pieces to their assigned ranks.
- `combine_dict(x)` gathers sparse dictionary-style updates that arise inside the bound-tightening routines.

Concentrating these operations in one module isolates the serialization logic from the algorithmic layers, which can then call them as if they were standard numerical primitives.

4.6.3 Data partitioning

Two partitioning patterns are used repeatedly. The routine `partition_co ncat(n)` assigns contiguous index blocks to ranks, which is cache-friendly when each rank loops over its own block. The routine `partition_shuf fle(n)` assigns indices in a round-robin pattern, which is preferable when per-index work is heterogeneous and load balancing is more important than

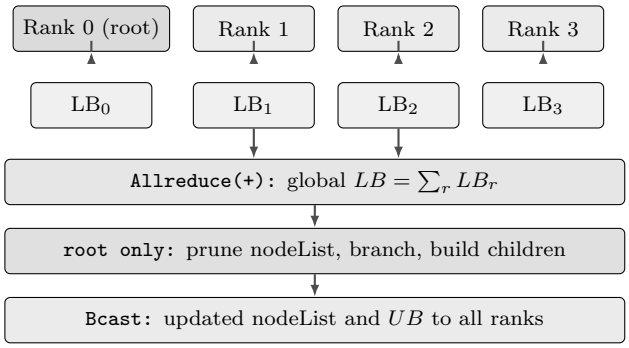

Figure 4.2: One iteration of the SPMD loop. Bound computations are parallelized across ranks, whereas node-list maintenance remains sequential on the root and is then broadcast to the remaining ranks.

locality. The file `src/group.jl` builds higher-level group bookkeeping on top of these primitives.

4.6.4 Parallel subgradient and load balancing

The Lagrangian dual is the most communication-intensive component of the search. Each rank evaluates its local contribution to the lower bound and its local subgradient, after which the relevant quantities are aggregated through `MPI.Allreduce`. The multiplier update is then performed on the root and broadcast back to all ranks. Figure 4.2 illustrates this pattern. Because the update depends only on global reductions, the parallel method is algorithmically identical to its serial counterpart up to floating-point reduction order.

4.7 Experimental Tooling, Computational Workflow, and Reproducibility

Beyond the core solver, the software includes a computational workflow that supports extension, experiment management, and reproducibility. This is the software counterpart of the computational study design discussed in the thesis.

4.7.1 Adding a new clustering variant

Because the orchestrator is the only routine that directly touches the global search state, extending the package to a new clustering model is a localized change. A typical implementation path consists of the following steps.

1. Declare a new subtype `Node_xxx` of `Node` in `src/Nodes.jl` containing whatever auxiliary fields the new lower bound requires.
2. Add a lower-bound implementation in `src/lb_functions/xxx_lb.jl` with the standard calling convention.
3. Add a feasible upper-bound heuristic in `src/ub_functions/xxx_ub.jl`.
4. Reuse the existing branching infrastructure when possible, and add a specialized tightening or probing routine only if the new model requires it.
5. Add the corresponding dispatch logic to `src/bb_function.jl` so that the orchestrator can construct the new node type and invoke the new components.
6. Extend the command-line parser so that the new variant can be called from batch experiments.

Listing 4.2 sketches the minimal lower-bound skeleton needed to satisfy the package interface.

```
module xxx_lb
using ..parallel
using ..group

export xxx_getLowerBound

function xxx_getLowerBound(X, k, node, UB)
    p, n = size(X)
    my_idx = group.get_optimized_local_indices(n, parallel.nprocs())
    LB_local = 0.0
    for s in my_idx
        LB_local += local_bound_contribution(X[:, s], k, node.lower, node.
            upper)
    end
    return parallel.sum(LB_local)
end
end
```

Listing 4.2: Minimal lower-bound skeleton for a new clustering variant.

The key point is that the routine does not communicate with MPI directly. It obtains its local sample indices from the grouping layer and relies on the parallel wrapper for aggregation. The same source code can therefore run in serial Julia, under `mpiexec`, or inside a job scheduler without modification.

4.7.2 Command-line interface and experiment driver

The script `test/test_run.jl` serves as the canonical experiment driver. It parses the main run parameters—algorithm, dataset, number of clusters, method, time limit, random seed, and output path—loads the requested data, calls `branch_bound` once, and records the resulting run summary. A typical invocation takes the form

```
mpiexec -n 16 julia --project=.
test/test_run.jl -a kmeans -d glass -k 3
   -m LD+SG -t 3600 -s 123
```

The same driver can be embedded in a SLURM batch script that enumerates the Cartesian product of algorithms, methods, datasets, and cluster counts. Since each experiment appends a single structured row to an aggregated CSV file, the outputs can be analyzed downstream by standard data-analysis tools without a custom parser.

4.8 Software Availability

The software accompanying this thesis is organized as a Julia project that contains the package source, experiment drivers, batch scripts, aggregated result tables, and the environment files needed to recreate the computational setup. The implementation targets Julia 1.9 or newer, requires a working MPI installation, and uses IBM ILOG CPLEX Optimization Studio (International Business Machines Corporation, 2022) for the mixed-integer subproblems discussed earlier in the thesis. A repository README documents installation, smoke tests, and representative invocations for the supported `(algo, method)` combinations listed in Table 4.2.

4.9 Summary

This chapter has translated the mathematical framework of the thesis into a modular software architecture. The resulting implementation separates global search control, model-specific bounds, domain reduction, and communication logic, while supporting large-scale parallel experiments through

MPI-based execution and a reproducible experiment workflow. Together, these software design choices make it possible to evaluate globally optimized clustering in a way that is both computationally scalable and methodologically transparent.

Table 4.1: Top-level module responsibilities in `src/`.

Module	**Responsibility**
`GO_Clustering.jl`	Top-level module that re-exports the main entry points for branching, parallel communication, grouping, and logging.
`bb_function.jl`	Complete B&B life cycle, including initialization, node selection, lower and upper bound evaluation, pruning, branching, and termination checks.
`Nodes.jl`	Abstract node definition together with the concrete node types for K-means, K-centers, and K-medoids.
`lb_functions/`	Lower-bound algorithms, including closed-form bounds, Lagrangian dual routines, subgradient updates, and adaptive grouping.
`ub_functions/`	Upper-bound heuristics, including Lloyd-style local search, PAM-based medoid improvement, and farthest-point-first initialization.
`bt_functions/`	Feasibility-based bound tightening and related domain-reduction routines.
`branch/`	Variable selection, symmetry breaking, and node splitting rules.
`probing/`	Explicit probing routines that test candidate sub-boxes and discard infeasible or dominated regions.
`opt_functions/`	JuMP formulations of local or grouped subproblems, with solver calls routed to the configured solver backends.
`parallel.jl`	MPI initialization, collective communication wrappers, and serialization support for Julia objects.
`group.jl`	Data and group partitioning across ranks, including the case where the number of processes exceeds the number of active groups.
`logging.jl`	Structured run records, text logs, and aggregated CSV output.

Table 4.2: Lower-bound methods exposed through the `method` keyword.

algo	method	File	Idea
`kmeans`	`CF`	`kmeans_lb.jl`	Closed-form bound obtained by clamping each sample to the current box and summing minimum squared distances.
`kmeans`	`SG`	`kmeans_lb.jl`	Adaptive sample grouping with a global optimization model per group.
`kmeans`	`LD+SG`	`kmeans_lb.jl`	Lagrangian decomposition over adaptive groups with subgradient updates.
`kcenters`	`CF`	`kcenters_lb.jl`	Per-sample farthest-from-box bound, aggregated through a maximum over samples.
`kcenters`	`FBBT`	`kcenters_lb.jl` and `kcenters_fbbt.jl`	Closed-form bounding combined with feasibility-based domain reduction.
`kmedoids`	`BB+Basic`	`kmedoids_lb.jl`	Analytic bound on feasible medoid candidates using current kernel regions.
`kmedoids`	`BB+LD`	`kmedoids_lb.jl`	Analytic lower bound strengthened by a Lagrangian dual and kernel-region certificates.

Chapter 5

Computational Experiments

This chapter reports computational experiments for the proposed clustering package `GO_Clustering.jl`. The experiments are carried out on Trillium, a parallel cluster hosted by SciNet at the University of Toronto and provided through the Digital Research Alliance of Canada (Alliance Canada, `alliancecan`). The compute nodes use AMD EPYC 9655 (Zen 5) processors at 2.6 GHz. The software targets Julia $\geq$ 1.6.7 and IBM ILOG CPLEX Optimization Studio 22.1.2 (International Business Machines Corporation, 2022).

5.1 K-means Results

We first test K-means on two small benchmark datasets, Iris (R. A. Fisher, 1936) and Seeds (Charytanowicz et al., 2010), together with the larger Hemicellulose dataset, with dimensions $(n, d) = (150, 4)$, $(210, 7)$, and $(1{,}955, 7)$, respectively. Here, n denotes the number of samples and d the number of attributes. The Hemicellulose instance is derived from the 1,955-point literature dataset compiled by Wang et al. (2022); in our experiments, each record is loaded through the package preprocessing pipeline as the seven-dimensional numeric feature representation used by `GO_Clustering.jl`, and the same normalization procedure described in Chapter 4 is applied before optimization. In this subsection, all experiments use $k = 3$. Here, BB denotes the branch-and-bound framework of Chapter 3. To stay consistent with the `method` keyword in Chapter 4, we label the exact variants as CF, SG, and LD+SG: CF uses the closed-form lower bound, SG uses the sample-grouping lower bound, and LD+SG uses the Lagrangian decomposition bound with sample grouping. LD+SG (20 cores) denotes the same method executed in parallel on 20 cores. For the heuristic baseline, `Clustering.jl` is run 100 times from random initial centers, and the corresponding worst, average, and best objective values are recorded. For the branch-and-bound variants, the objective column reports the final upper

bound. Table 5.1 also reports the number of processed nodes, the final relative gap, and the reported wall-clock time when available.

Three points are immediate. First, on Iris and Seeds, all branch-and-bound variants recover the best objective found over the 100 heuristic runs. Second, LD+SG is much more effective than CF and SG in terms of node count, and the 20-core version reduces wall-clock time on Iris; on Seeds, it attains the smallest reported final gap among the listed K-means variants. Third, on the larger Hemicellulose instance, the best heuristic objective is again matched, but certification quality varies substantially across methods; among the tested variants, LD+SG with 20 cores gives the smallest reported final gap.

5.2 K-centers Results

We next test K-centers on the same two small datasets, Iris (R. A. Fisher, 1936) and Seeds (Charytanowicz et al., 2010), together with the larger Hemicellulose dataset, again with $k = 3$. Here, FFT denotes the Farthest First Traversal heuristic introduced in Section 3.3.2. We compare FFT, a CPLEX-based baseline, CF, and FBBT, where FBBT denotes feasibility-based bound tightening. Table 5.2 reports the objective value, the number of processed nodes, the final relative gap, and runtime.

The pattern is clear. On Iris and Seeds, FBBT matches the best reported objective while dramatically reducing node counts relative to both CPLEX and CF. On Hemicellulose, the advantage is even stronger: FBBT attains a substantially smaller objective than the other reported methods and reduces the final gap to at most 0.1% with only 11 processed nodes.

5.3 K-medoids Results

We finally test K-medoids on the same two small datasets, Iris (R. A. Fisher, 1936) and Seeds (Charytanowicz et al., 2010), together with the larger Hemicellulose dataset, again with $k = 3$. We report a heuristic baseline, a direct CPLEX solve of the MILP, and the two branch-and-bound variants used in the software interface, BB+Basic and BB+LD. Table 5.3 reports the final upper bound, the number of processed nodes, the final relative gap, and runtime.

The K-medoids results reinforce the value of stronger lower bounds. On

Iris, the heuristic baseline is slightly suboptimal, whereas all exact methods recover the better objective value 83.91. Among the exact methods, CPLEX is fastest on this instance, while BB+LD reduces the search to only 25 nodes, compared with 2.0×10^5 for BB+Basic.

On Seeds, the heuristic baseline already matches the best reported objective, but the certification effort still differs sharply across methods. BB+LD reduces the node count from 2.4×10^5 under BB+Basic to only 9 nodes, and cuts the corresponding runtime from 270 to 78 seconds.

The largest contrast appears on Hemicellulose. All methods report the same incumbent value, but their ability to certify it differs substantially. BB+Basic requires 1.9×10^6 nodes yet finishes with a 4.32% gap after four hours. By contrast, BB+LD closes the gap to at most 0.1% in 93 seconds using only 63 nodes, substantially outperforming both BB+Basic and the direct CPLEX solve.

Table 5.1: K-means results on Iris, Seeds, and Hemicellulose for $k = 3$. For `Clustering.jl`, the reported objective values are the worst, average, and best outcomes.

Method	**Obj.**	**Nodes**	**Gap (%)**	**Time**
Iris ($n = 150, d = 4$)				
`Clustering.jl` worst	145.45	–	–	–
`Clustering.jl` average	85.91	–	–	–
`Clustering.jl` best	78.85	–	–	–
CF	78.85	3.30×10^5	53.21	–
SG	78.85	883	1.23	–
LD+SG	78.85	31	0.10	1.0 h
LD+SG (20 cores)	78.85	31	0.10	0.5 h
Seeds ($n = 210, d = 7$)				
`Clustering.jl` worst	916.21	–	–	–
`Clustering.jl` average	591.46	–	–	–
`Clustering.jl` best	587.32	–	–	–
CF	587.32	3.28×10^5	83.21	–
SG	587.32	440	9.27	–
LD+SG	587.32	44	0.23	–
LD+SG (20 cores)	587.32	92	0.10	0.8 h
Hemicellulose ($n = 1{,}955, d = 7$)				
`Clustering.jl` worst	1.698×10^7	–	–	–
`Clustering.jl` average	1.020×10^7	–	–	–
`Clustering.jl` best	9.750×10^6	–	–	–
CF	9.750×10^6	3.27×10^5	59.48	–
SG	9.750×10^6	74	21.75	–
LD+SG	9.750×10^6	4	39.38	–
LD+SG (20 cores)	9.750×10^6	112	2.23	–

Table 5.2: K-centers results on Iris, Seeds, and Hemicellulose for $k = 3$.

Method	**Obj.**	**Nodes**	**Gap (%)**	**Time**
Iris ($n = 150, d = 4$)				
FFT	2.65	–	–	–
CPLEX	2.04	1.2×10^5	≤ 0.1	46 s
CF	2.04	1.3×10^4	≤ 0.1	17 s
FBBT	2.04	1	≤ 0.1	12 s
Seeds ($n = 210, d = 7$)				
FFT	13.17	–	–	–
CPLEX	10.44	1.2×10^6	≤ 0.1	542 s
CF	10.44	7.2×10^3	≤ 0.1	17 s
FBBT	10.44	21	≤ 0.1	13 s
Hemicellulose ($n = 1{,}955, d = 7$)				
FFT	1.06×10^5	–	–	–
CPLEX	4.08×10^5	1.4×10^5	100.0	4 h
CF	4.08×10^5	1.4×10^5	50.0	4 h
FBBT	6.49×10^4	11	≤ 0.1	14 s

Table 5.3: K-medoids results on Iris, Seeds, and Hemicellulose for $k = 3$.

Method	**Obj.**	**Nodes**	**Gap (%)**	**Time**
Iris ($n = 150, d = 4$)				
Heuristic	84.63	–	–	–
CPLEX	83.91	1	≤ 0.1	40 s
BB+Basic	83.91	2.0×10^5	≤ 0.1	100 s
BB+LD	83.91	25	≤ 0.1	98 s
Seeds ($n = 210, d = 7$)				
Heuristic	598.29	–	–	–
CPLEX	598.29	1	≤ 0.1	48 s
BB+Basic	598.29	2.4×10^5	≤ 0.1	270 s
BB+LD	598.29	9	≤ 0.1	78 s
Hemicellulose ($n = 1{,}955, d = 7$)				
Heuristic	9.91×10^6	–	–	–
CPLEX	9.91×10^6	1	≤ 0.1	0.5h
BB+Basic	9.91×10^6	1.9×10^6	4.32	4 h
BB+LD	9.91×10^6	63	≤ 0.1	93 s

Chapter 6

Conclusion

This thesis set out to develop a unified global-optimisation framework for centroid-based clustering that is mathematically rigorous, computationally scalable, and reproducible. Within the scope studied here, that goal has been achieved. K-means, K-centers, and K-medoids are brought together through a consistent modelling perspective, a common branch-and-bound framework, and a shared Julia implementation. In a literature where exact methods for these models are often reported separately, the thesis contributes an integrated framework that makes formulation, algorithmic comparison, and certification more transparent. Its significance lies not only in individual exact methods, but in showing that globally optimised clustering can be studied as a coherent optimisation and software problem.

The main strengths of the thesis are therefore unified modelling across three canonical objectives, certificate-producing algorithms, and reproducible software. The main limitations are equally clear: the study is confined to the centroid-based models and computational settings treated here, and performance remains strongly dependent on lower-bound quality and problem structure.

These findings are most relevant where clustering is used to build benchmarks, compare algorithms, design representative partitions, or support downstream analysis and optimality guarantees matter. The most natural next steps are to strengthen model-specific bounds and branching rules, broaden the data and distance settings, enlarge the benchmark set, and improve parallel search and software interoperability. Overall, the thesis provides a practical and methodological foundation for future research on exact and certifiable clustering, and this integrated perspective is its main contribution to the broader discipline.

References

Aloise, D., Deshpande, A., Hansen, P., & Popat, P. (2009). Np-hardness of euclidean sum-of-squares clustering. *Machine learning*, *75*(2), 245–248.

Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1027–1035.

Barahona, F., & Anbil, R. (2000). The volume algorithm: Producing primal solutions with a subgradient method. *Mathematical Programming*, *87*(3), 385–399.

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Blaom, A. D., Kiraly, F., Lienart, T., Simillides, Y., Arenas, D., & Vollmer, S. J. (2020). Mlj: A julia package for composable machine learning. *arXiv preprint arXiv:2007.12285*.

Bottou, L., & Bengio, Y. (1994). Convergence properties of the K-means algorithms. *Advances in neural information processing systems*, *7*, 585–592.

Cao, Y., & Zavala, V. M. (2019). A scalable global optimization algorithm for stochastic nonlinear programs. *Journal of Global Optimization*, *75*(2), 393–416.

Celebi, M. E., Kingravi, H. A., & Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert systems with applications*, *40*(1), 200–210.

Charytanowicz, M., Niewczas, J., Kulczycki, P., Kowalski, P. A., Łukasik, S., & Żak, S. (2010). Complete gradient clustering algorithm for features analysis of x-ray images. In *Information technologies in biomedicine: Volume 2* (pp. 15–24). Springer.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. *kdd*, *96*(34), 226–231.

Fisher, M. L. (1981). The lagrangian relaxation method for solving integer programming problems. *Management science*, *27*(1), 1–18.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, *7*(2), 179–188.

Frey, B. J., & Dueck, D. (2007). Clustering by passing messages between data points. *science*, *315*(5814), 972–976.

Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, *38*, 293–306. https://doi.org/10.1016/0304-3975(85)90224-5

Hua, K., Cao, Y., & Shi, M. (2021). Global optimization of K-means clustering via reduced-space branch-and-bound with Lagrangian decomposition. *Proceedings of the 38th international conference on machine learning*, *139*, 4357–4366.

International Business Machines Corporation. (2022). *IBM ILOG CPLEX Optimization Studio: User's manual for cplex*. Version 22.1.2. Retrieved April 8, 2026, from https://www.ibm.com/docs/en/icos/22.1.2

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern recognition letters*, *31*(8), 651–666.

JuliaStats Contributors. (2025). *Clustering.jl: Data clustering in Julia* (Version 0.15.8). Retrieved April 2026, from https://github.com/JuliaStats/Clustering.jl

Kaufman, L., & Rousseeuw, P. J. (1990). *Finding groups in data: An introduction to cluster analysis*. John Wiley & Sons. https://doi.org/10.1002/9780470316801

Kratochvíl, M., Hunewald, O., Heirendt, L., Verissimo, V., Vondrášek, J., Satagopam, V. P., Schneider, R., Trefois, C., & Ollert, M. (2020). Gigasom.jl: High-performance clustering and visualization of huge cytometry datasets. *GigaScience*, *9*(11), giaa127.

Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, *28*(2), 129–137. https://doi.org/10.1109/TIT.1982.1056489

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, *1*, 281–297.

Mahajan, M., Nimbhorkar, P., & Varadarajan, K. (2012). The planar k-means problem is np-hard. *Theoretical computer science*, *442*, 13–21.

Park, H.-S., & Jun, C.-H. (2009). A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, *36*(2), 3336–3341.

Ren, J., Hua, K., & Cao, Y. (2022). Global optimal K-medoids clustering of one million samples. *Advances in Neural Information Processing Systems*, *35*.

Ren, J., You, N., Hua, K., Ji, C., & Cao, Y. (2025). A global optimization algorithm for k-center clustering of one billion samples. *Management Science*.

Rockafellar, R. T., & Wets, R. J.-B. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of operations research*, *16*(1), 119–147.

Schubert, E., & Rousseeuw, P. J. (2021). Fast and eager K-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms. *Information Systems*, *101*, 101804. https://doi.org/10.1016/j.is.2021.101804

Shi, M., Hua, K., Ren, J., & Cao, Y. (2022). Global optimization of K-center clustering. *Proceedings of the 39th international conference on machine learning*, *162*, 20094–20118.

Velmurugan, T., & Santhanam, T. (2010). Computational complexity between k-means and k-medoids clustering algorithms for normal and uniform distributions of data points. *Journal of computer science*, *6*(3), 363.

Wang, E., Ballachay, R., Cai, G., Cao, Y., & Trajano, H. L. (2022). Predicting xylose yield from prehydrolysis of hardwoods: A machine learning approach. *Frontiers in Chemical Engineering*, *4*, 994428.

Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., et al. (2008). Top 10 algorithms in data mining. *Knowledge and information systems*, *14*(1), 1–37.