How to Use K-means for Big Data Clustering?

Ravil Mussabayev

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2024

Reading Committee:
Gunther Uhlmann, Chair
Simon Du
Bamdad Hosseini

Program Authorized to Offer Degree:
Mathematics

University of Washington

Abstract

How to Use K-means for Big Data Clustering?

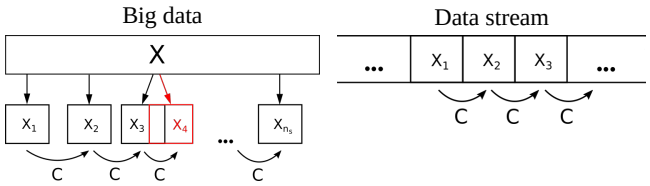
Ravil Mussabayev

Chair of the Supervisory Committee:
Gunther Uhlmann
Department of Mathematics

K-means plays a vital role in data mining, being the simplest and most widely used algorithm under the Euclidean Minimum Sum-of-Squares Clustering (MSSC) model. However, its performance drastically drops when applied to vast amounts of data. Therefore, it is crucial to improve K-means by scaling it to big data using as few of the following computational resources as possible: data, time, and algorithmic ingredients. We introduce a novel parallel scheme that leverages K-means and K-means++ algorithms for big data clustering, offering a "true big data" algorithm that excels in both solution quality and runtime, surpassing classical and recent state-of-the-art MSSC approaches. The new approach naturally implements global search by decomposing the MSSC problem without using additional metaheuristics. On the other hand, this approach can be generalized to a novel metaheuristic, providing fresh perspectives for creating new powerful optimization heuristics. This work shows that data decomposition is the basic approach to solve the big data clustering problem. The empirical success of the new algorithm and its derivatives allowed us to challenge the common belief that more data is required to obtain a good clustering solution. Moreover, the present work questions the established trend that more sophisticated hybrid approaches and algorithms are required to obtain a better clustering solution.

Big-means: K-means for Big Data Clustering

function **Big-means**(X, k, s):
C ← {∅$_1$, ..., ∅$_k$}
f* = ∞
while get_time() < t_{max}:
  P ← uniform sample of size s from X
  C' ← reinitialize degenerate in C by K-means++
  C'' ← K-means(P, C')
  if f* > f(C'', P):
    C ← C''
    f* ← f(C'', P)
return C



✓ Good quality results with incomplete use of all available data
✓ Configurable trade-off between resulting quality and speed
✓ Effectiveness in clustering small, large, and big data
✓ Ability of stream and parallel processing
✓ Improvement of clustering results with increase in amount of processed data

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

HEURISTIC: a practical problem-solving approach or algorithm that, while not guaranteed to be optimal or perfect, is sufficient for reaching an immediate, short-term goal or approximation.

METAHEURISTIC: a higher-level heuristic designed to select and modify simpler heuristics, with the goal of efficiently exploring a solution space and finding near-optimal or optimal solutions.

"LESS IS MORE" (LIMA) PRINCIPLE: an approach whose main idea is in using as few ingredients as possible to provide the best possible outcome.

ALGORITHMIC INGREDIENTS: a collection of typical high-level operations that serve as common components in various search algorithms designed to solve the specific problem at hand.

LIMA NUMBER: the cardinality of the set of ingredients employed by a method to solve an optimization problem on a dataset.

LIMA DOMINANCE: a criterion in which an algorithm is considered superior if it demonstrates better or equal performance in terms of accuracy and speed, while also being simpler (having fewer components or steps) compared to another algorithm, with at least one of these factors being strictly better.

"TRUE BIG DATA" CLUSTERING ALGORITHM: an algorithm that achieves the optimal balance between simplicity, resulting quality, and convergence speed; performs a global search for solutions in big data contexts without relying on complex global optimization metaheuristics; exhibits scalability through features like configurable quality-speed trade-offs, effectiveness across all data sizes, ability to improve resulting quality with more data, possibility of obtaining acceptable results with incomplete use of all data, as well as capabilities for parallel, distributed, and stream processing.

TRADITIONAL CLUSTERING ALGORITHM: an algorithm, like K-means [HJ82] and its variants, that is widely recognized for its simplicity, effectiveness, and extensive study,

making it a standard approach for solving the clustering problem despite limitations in dataset size.

ALTERNATIVE CLUSTERING ALGORITHM: an algorithm, such as MDEClust [MS21] and HG-means [GV19], that is complex, hybrid, and designed to improve solution quality for the clustering problem, but often lags in time efficiency and faces challenges in user adoption due to its intricacy and implementation difficulties.

SCALABILITY: the ability to configure the quality-speed trade-off, improve clustering quality with increased data, handle datasets of varying sizes, produce acceptable results without using all available data, facilitate parallel and distributed processing, and process incoming data streams [Mus+23].

ACKNOWLEDGMENTS

I am profoundly grateful to my advisor, Prof. Gunther Uhlmann, whose unwavering support and invaluable wisdom have been the cornerstone of my PhD journey. His guidance and encouragement have been instrumental in shaping my academic and professional growth.

My heartfelt thanks go to my brother (and professor) Rustam Mussabayev and Prof. Nenad Mladenovic. Working alongside such esteemed experts has been an enriching experience. Their deep insights and innovative ideas have significantly contributed to my research and understanding of the field.

To my beloved mother, Natalya, I offer my deepest gratitude for her unshakeable faith in me. Her boundless love and care have been a guiding light, providing support and inspiration at every turn of my academic journey.

I also extend my sincere appreciation to all the professors and fellow students who have been a part of my academic path, both in my undergraduate and graduate studies. Their diverse viewpoints, collaboration, and camaraderie have greatly enhanced my learning experience and personal growth.

This journey would not have been possible without the collective support and guidance of each of these remarkable individuals. I am eternally thankful for their contributions to my academic and personal development.

DEDICATION

to my dear mom, Natalya

Chapter 1

INTRODUCTION

1.1 Problem formulation

Clustering is a foundational task that groups similar objects within a set, revealing inherent structures and relationships. Cluster analysis methods have proven to be a powerful tool for data mining, solving the problem of unsupervised classification of patterns. This problem is challenged by the rapid growth of digital data and has applications in many domains, such as image and video analysis [YYL98], customer segmentation [Che+18], information retrieval [Dje+21], anomaly detection [Tu+20], pattern recognition and classification [RRR13], vector quantization and data compression [Yin+15], natural language processing [Alg+19], bioinformatics [RRR13], gene expression analysis [JTZ04], network and traffic analysis [DWV08], time series analysis [Rak+12], medical diagnosis [Mit+21], social media analysis [ZZ11].

The goal of cluster analysis is to organize a collection of objects into subsets, called clusters, by similarity. This is achieved by maximizing intra-cluster similarity and/or minimizing inter-cluster dissimilarity [Bag08]. These two criteria are called homogeneity and separation, respectively.

There are many different methods for cluster analysis. Let us define a notation

$$\mathbb{N}_i = \{1, 2, \ldots, i\}.$$

Then, the hard unconstrained partition clustering problem is to find subsets $X_j \subset X$, where $j \in \mathbb{N}_k$, satisfying the following criteria:

1. $X_j \neq \varnothing,\ \forall j \in \mathbb{N}_k$;

2. $X_j \cap X_l = \varnothing,\ \forall j, l \in \mathbb{N}_k$ such that $j \neq l$;

3. $X = \bigcup_{j=1}^{k} X_j$.

The subsets $\{X_j\}_{j=1}^{k}$ are called clusters.

In this work, we consider the Minimum Sum-of-Squares Clustering (MSSC) problem. It is a fundamental problem in cluster analysis, being one of the most extensively studied [Alo+09]. The MSSC problem can be formulated as follows.

Given a set of m data points $X = \{x_1, \ldots, x_m\}$ in the Euclidean space $\mathbb{R}^n$, the goal of MSSC is to find k cluster centers (centroids) $C = (c_1, \ldots, c_k) \in \mathbb{R}^{n \times k}$ that minimize the sum of squared distances from each data point x_i to its nearest cluster center c_j:

$$\min_{C} \quad f(C, X) = \sum_{i=1}^{m} \min_{j=1,\ldots,k} \|x_i - c_j\|^2 \tag{1.1}$$

where $\|\cdot\|$ stands for the Euclidean norm. Equation (1.1) is the objective function, which is called the sum-of-squared distances. In the MSSC problem, we assume that each cluster X_j can be uniquely identified by its centroid $c_j \in \mathbb{R}^n$ for every $j \in \mathbb{N}_k$.

For any k and m, the MSSC problem poses a significant challenge in producing high-quality solutions due to its NP-hard nature [Alo+09], which is exacerbated by big data contexts.

MSSC is a global optimization problem, whose objective is to minimize (1.1) by properly dividing the dataset into a given number of clusters [HJ97]. For the MSSC problem, homogeneity and separation are dual objectives. In other words, making groups as similar as possible (homogeneity) is equivalent to making groups as distinct as possible (separation). The major advantage of MSSC is that by minimizing the single objective, as expressed in equation (1.1), both of these dual goals are simultaneously attained. As a result, clustering algorithms are evaluated based on their accuracy, as measured by the objective function in (1.1), which serves as a critical performance metric.

When dealing with big data, where the number of data points is unbounded, i.e., $|X| =$

$m = \infty$, formulation (1.1) gives rise to the Minimum Sum-of-Squares Clustering of Infinitely Tall Data (MSSC-ITD) problem, which is one of the key contributions of our work. This problem makes traditional clustering methods unfeasible. The MSSC-ITD problem is a novel formulation that we have introduced in this paper, and our proposed algorithm is the first to provide an efficient solution to this challenge. In particular, few clustering algorithms exist that can address this problem, and even fewer can perform a global search in such conditions. Our approach fills this gap, providing a robust and efficient solution to the MSSC-ITD problem.

Optimal solutions to the MSSC problem are known to satisfy at least two necessary conditions [GV19]. These properties can be formulated as follows:

1. In any optimal MSSC solution, for each $j \in \mathbb{N}_k$, the position of center c_j coincides with the centroid of the points belonging to X_k:

$$c_j = \frac{1}{|X_k|} \sum_{x_i \in X_k} x_i \tag{1.2}$$

2. In any optimal MSSC solution, each entity x_i is associated with the closest centroid $c_{k_{min}(i)}$ such that:

$$k_{min}(i) = \arg\min_{j \in \mathbb{N}_k} \|x_i - c_j\| \tag{1.3}$$

These two properties are fundamental to understanding the behavior and properties of the MSSC algorithms.

The next section motivates the present work by describing the unique challenges the MSSC problem poses in the context of big data.

1.2 Motivation and key challenges

1.2.1 Effectiveness and global optimization issue

Viewed as a global optimization problem, MSSC aims to divide a dataset into clusters, optimizing a single objective detailed in (1.1) that inherently enhances within-cluster similarity and between-cluster dissimilarity. This optimization criterion is fundamental in assessing the performance of clustering algorithms, with global minimizers offering a more precise reflection of a dataset's inherent clustering structure [GV19]. However, the search for global minimizers is hindered by the objective function's significant non-convexity, making MSSC a complex yet crucial problem in cluster analysis. The non-convex landscape of the MSSC objective function becomes dramatically more complex as the number of data points increases, which is especially characteristic to big data conditions.

To address this MSSC challenge, which is exacerbated by big data contexts, several approaches have been proposed in the literature to explore the solution space and locate global minimizers which include, but are not limited to: gradient-based optimization techniques [KBT18], stochastic optimization algorithms [Mus+23], metaheuristic search strategies [GV19; HM01], and hybrid approaches [MS21]. Gradient-based techniques [KBT18] have a focus on providing a fast convergence to local minimizers but may get trapped in poor solutions due to the non-convex nature of the objective function. On the other hand, stochastic optimization algorithms [Mus+23] incorporate randomness in the search process to escape local minima and explore a broader solution space. Metaheuristic search strategies [GV19; HM01] aim to balance exploration and exploitation in the search process. Hybrid approaches [MS21] are frequently used, as they can combine multiple different methods with the goal of not only leveraging their advantages but also acquiring new qualitative properties.

Despite the proposition of these numerous optimization techniques aimed at overcoming the high non-convexity challenge of MSSC, each method presents its unique advantages and limitations, indicating the absence of a universally applicable solution, especially in big data environments. Consequently, this underscores the need for continued investigation to devise

methodologies that are both more effective and efficient in identifying global minimizers, particularly within the intersection of the MSSC area with the expansive field of big data.

Yet another prominent difficulty of the MSSC model is that the number of clusters is not known a priori [Bag08]. However, we do not consider this problem in the current study. Instead, we regard it as an interesting and promising future research direction.

1.2.2 Big data and its challenges

Although the notions of large and big data are frequently used in the literature, there are no clear and universally accepted definitions of these notions. Generally, big data refers to the vast amount of information that is generated and collected in various forms, such as text, images, videos, and sensor readings.

Big data is characterized by the following three main "VVV" properties: large volume, velocity, and variety [DGG15]. Volume refers to the amount of generated and stored data. The size of big data is usually larger than terabytes and petabytes [SS13]. Velocity refers to the speed at which data is generated and processed. Big data is often generated in real time and at high speed. For instance, the American retail corporation Walmart handles more than 1 million customer transactions every hour. Variety refers to the type and nature of data. Big data can be either structured or unstructured (raw). The vast majority of big data is unstructured and cannot be easily stored in a relational database. Some examples of the variety of big data include photos, webpages, audio, video, posts on social networks, mobile data, etc.

To solve the MSSC problem, we consider only structured data, which is represented in the vector form. From the perspective of vectorization, structured big data can be of three main types:

- with a large number of vector representations (volume);

- with a large number of features (variety);

- mixed type (volume + variety).

In this work, we consider only the type of big data with many vector representations and a relatively small number of features (vector components). This choice is mainly due to the use of Euclidean distance in the MSSC problem, which is not an optimal metric for measuring distances between high-dimensional vectors [AHK01]. Addressing problems related to other types of big data is a subject for future research.

The properties of big data make it difficult to manage, process, and analyze big datasets using traditional data processing tools and techniques. The precise definition of "big data" can vary depending on the context and the resources available for processing and analysis. For humans, "big data" refers to datasets that are impossible to process or analyze manually. Accordingly, for computers, big data represents a scenario where data cannot be processed using standard computational and algorithmic resources. Thus, we define "big data" as datasets that are excessively large, complex, or diverse, rendering traditional processing methods impractical on an average computer due to limited computing resources or prohibitive time costs [Mus+23]. In other scenarios under these conditions, data processing may either encounter no difficulties or face technical challenges yet remain feasible. Accordingly, we define "small data" and "large data" based on these two situations. Consequently, in our work, big data refers to datasets of such immense size that their processing either presents substantial technical challenges or becomes infeasible with conventional methods and standard computing resources.

Considering these challenges, it is essential to develop and implement optimized algorithms for a wide spectrum of downstream tasks. These algorithms should have enhanced efficiency and be able to process large datasets using relatively limited computing resources. Alternatively, they should be scalable to accommodate additional computing resources.

In Figure 1.1, a publication growth graph shows the increase in the number of publications on the topic of big data clustering over time, highlighting its growing relevance. The statistics was collected via the Web of Science citation network by searching the Web of Science Core

Collection by keywords “clustering big data”.

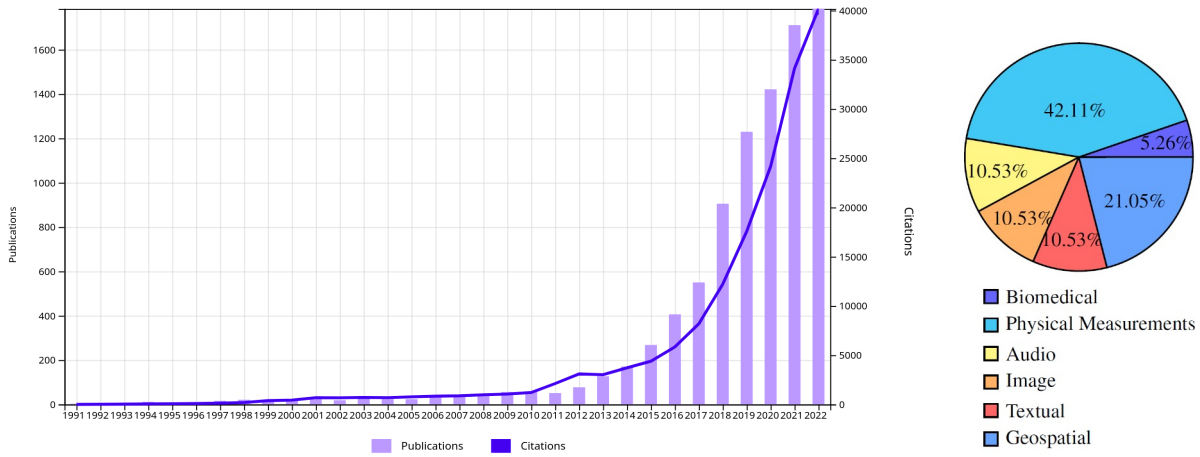


Figure 1.1: Trends in big data publications and citations over the years alongside common data types used in big data clustering

The data types most commonly used for big data clustering are represented on the right side of Figure 1.1. Specifically, the pie chart shows the distribution of different types of data (e.g., text, images, videos, sensor readings) that are frequently used for big data clustering [MHE21b], including the current study. The statistics for constructing the pie chart was based on the sample of the datasets used in our experimental section. The chart highlights the diversity of data that big data clustering techniques can handle.

The MSSC problem has been well studied theoretically [CYY20]. Numerous approximate [CPL20] and a number of exact [PSW22] algorithms were proposed. However, most of these algorithms either have a linear or superlinear dependence of the required computational costs on the dataset size. Hence, most of existing approaches are only suitable for relatively small datasets. The NP-hardness of MSSC [Alo+09] and the size of practical datasets explain why most MSSC algorithms are heuristics, designed to produce an approximate solution in a reasonable computational time [GV19]. Nowadays, much of the research in this field is directed toward developing effective methods for solving the NP-hard MSSC problem [Alo+09] in real practical conditions of large and big datasets [KBT18], where most

of the classical methods have shown a lack of efficiency.

1.2.3 Efficiency and simplicity

In designing advanced specialized algorithms for big data, it is vital to prioritize simplicity by utilizing the minimal number of algorithmic components to achieve the best possible results [MPP22]. Algorithmic ingredients refer to a collection of typical high-level operations that serve as common components in various search algorithms designed to solve the specific problem at hand. Often, the efficiency and potential popularity of the heuristic are often directly linked to its simplicity, as it helps reduce computational and time expenses. This emphasis on simplicity becomes particularly significant in the context of big data processing.

The concept of simplicity, within the context of constructing optimization algorithms, can be rigorously formalized using mathematical principles. This concept, commonly referred to as the "less is more" principle, is thoroughly explored in Section 2.6 of the background chapter (Chapter 2).

In our study, we evaluate the performance of various clustering techniques on a large number of benchmark datasets, comparing them according to the dominance criterion provided by the "less is more" approach (LIMA), i.e., simultaneously along the dimensions of speed, clustering quality, and simplicity.

The LIMA dominance criterion, as established by Brimberg et al. [Bri+23], defines a comparative relation among a collection of algorithms. Specifically, it asserts that an algorithm is considered superior to another if it achieves equal or better performance in terms of average accuracy, execution speed, and simplicity simultaneously. Here, simplicity is quantified by the cardinality of the set of used algorithmic components, and it is required that the superior algorithm strictly outperforms the other in at least one of these dimensions. The aspect of simplicity becomes critical in the context of big data. A comprehensive discussion on the LIMA dominance criterion is provided in Section 2.7 of the background chapter (Chapter 2).

The K-means algorithm (Lloyd's algorithm [HJ82]) is widely accepted as one of the simplest and most efficient clustering algorithms, which naturally minimizes the MSSC cri-

terion (1.1), operating favorably with respect to the LIMA dominance criterion. It can be considered a classic method for solving the MSSC problem [CYY20]. When applied to small or large data, K-means can find high-quality local solutions in reasonable time [Jai10]. K-means can be regarded as a universal clustering algorithm that is often used as a local search in more sophisticated clustering approaches.

While K-means offers a fast local search within the solution space of the MSSC problem, it is not directly applicable to big data due to its time complexity of $\mathcal{O}(m \cdot n \cdot k)$. Making a full pass through the dataset is necessary for K-means, which becomes prohibitive under big data conditions. However, K-means is highly adaptable, allowing optimization at nearly every step for big data processing and offering flexible integration with advanced clustering metaheuristics. Moreover, K-means can still be effectively utilized to accelerate other clustering models, such as density-based clustering [BCL+17] and spectral clustering [Fil+08]. For a more detailed discussion on K-means, its steps, and opportunities for their optimization, we refer the reader to Chapter 2.

For a comprehensive review of available K-means-like methods and other approaches for solving the MSSC problem in big data conditions, see Chapter 5 and Section 3.2 of Chapter 3. Finding novel effective, efficient and yet simple ways to use K-means for clustering big data is the main goal of the current study.

1.2.4 Summary

Big data is widely viewed as a challenge for most standard algorithms and alternative heuristics to overcome. However, it can also be seen as an advantage, offering opportunities to enhance clustering results. Therefore, there is a significant need for new clustering algorithms tailored to big data, which are not only relatively simple and effective for processing large datasets, but also capable of leveraging big data as an advantage to improve clustering results. These algorithms should achieve a balance between simplicity, result quality, and convergence speed, while performing a global search for the optimal solution without relying on the established complex global optimization metaheuristics. The algorithms we propose in

this dissertation, which utilize parallel processing and intelligent input data sample selection, are designed to bridge this gap.

1.3 Methodology and scientific novelty

In this work, we develop three novel clustering algorithms: Big-means (along with its optimized parallel versions, called HPClust), BigVNSClust, and BigOptimaS3. They are designed to tackle the challenges described above. Additionally, we distill their shared concepts into a new metaheuristic, Variable Landscape Search (VLS), offering both an advancement and a fresh perspective in the field of optimization metaheuristics.

We develop new algorithms and rigorously assess them through comprehensive experimental analysis, confirming their effectiveness and efficiency. The experiments utilize a broad and diverse range of real-world datasets. The proposed algorithms surpass the state of the art in accuracy and time for small, large, and big data. Initially, in Chapter 3, we introduce Big-means, a novel K-means-based algorithm that enables effective and efficient big data clustering through the decomposition principle. Next, we optimize the obtained Big-means algorithm along the dimension of parallelization strategies and high-performance computing in Chapter 4, offering HPClust. Then, in Chapter 6, we improve the Big-means approach using the well-known VNS metaheuristic [MH97], resulting in the BigVNSClust algorithm. Additionally, we substantially enhance the proposed approach by incorporating parallel competitive stochastic sample size optimization. The obtained novel algorithm is called BigOptimaS3 and detailed in Chapter 7. Finally, in Chapter 8, we generalize the core principles of all the proposed methods in a versatile optimization metaheuristic, Variable Landscape Search (VLS), providing a solid theoretical basis for creating many more new advanced optimization algorithms beyond the scope of clustering.

Also, our research in Chapter 5 includes a thorough review of prominent big data clustering algorithms available in the literature, putting special emphasis on various techniques used for optimizing K-means clustering and other clustering approaches related to MSSC in big data environments. In that chapter, we evaluate their strengths and weaknesses, identifying

prevailing global challenges that need addressing. Also, we show that different techniques are more suitable for different types of datasets and provide insights into the trade-offs between speed and accuracy in optimizing K-means for big data. Additionally, Chapter 5 offers a comprehensive guide for practitioners and researchers on how to optimize K-means for big data applications.

Throughout our study, the primary focus is on answering the set of main research questions (RQs). Answers to them serve as a foundation for this dissertation. These RQs encapsulate the core theses of our research and are formulated as follows:

- RQ 1: What are the limitations of traditional and alternative clustering algorithms in analyzing big datasets? How can the traditional K-means algorithm be optimized in general and specifically for big data?

- RQ 2: Is there a clustering method that concurrently optimizes key criteria such as accuracy, speed, and simplicity (as per LIMA dominance [Bri+23]) when applied to big data? If not, can a novel approach be developed and empirically validated to maximize these criteria in big data clustering?

- RQ 3: Does the widely accepted belief that utilizing every data point in a dataset is essential for attaining a satisfactory clustering solution hold true?

- RQ 4: Does the prevailing trend of relying on more complex hybrid approaches hold true in the quest for improved clustering solutions?

- RQ 5: Based on the above findings, can we establish the precise definition of a "true big data" clustering algorithm?

Additionally, we aim at addressing a number of secondary RQs:

- RQ 6: How can a "true big data" clustering algorithm be optimized to make the most effective use of the modern high-performance computing (HPC) technologies?

- RQ 7: How can a "true big data" clustering algorithm be further enhanced by incorporating modern metaheuristic optimization techniques?

- RQ 8: Could the accuracy and efficiency of Big-means be further boosted by applying stochastic sample size optimization within a competitive parallel framework?

- RQ 9: Can a solid theoretical foundation be established for the proposed algorithms in the form of a metaheuristic that is comprehensive, practically applicable, and encompasses other existing metaheuristics in the literature?

The main contributions presented this dissertation are the following:

- Big-means is proposed as an innovative parallel clustering method for big data, built upon the foundations of the K-means and K-means++ algorithms (Chapter 3). This new approach employs dataset decomposition to efficiently discover high-quality clustering solutions with substantially less data compared to previous methods. Adhering to the principle of "the more data, the better", our algorithm demonstrates increasing advantages over other algorithms as the size of the dataset being analyzed grows larger;

- The Big-means algorithm demonstrates state-of-the-art performance on real-world datasets of various sizes, from small to large and big data, excelling in both accuracy and time efficiency. Its superiority over all competing algorithms is empirically established. When these empirical findings are combined with its advantageous theoretical complexity, Big-means is proven to outperform all competing algorithms in terms of the LIMA dominance relation (LIMA number [Bri+23]);

- Criteria for "true big data" clustering algorithms are proposed;

- An extensive evaluation of different parallelization methods for big data clustering is conducted, leading to the empirical identification of the optimal strategy and a new state-of-the-art big data clustering algorithm, HPClust (Chapter 4);

- An in-depth analysis of various K-means-like clustering methods and other MSSC techniques applied in big data scenarios is performed (Chapter 5), marking the first time the LIMA dominance criterion [Bri+23] is used in reviewing and assessing big data clustering algorithms. This review is augmented by thorough experiments across a broad spectrum of real-world datasets, with the outcomes interpreted through the innovative lens of the LIMA dominance criterion;

- We are the first to introduce practical guidelines and a flowchart, serving as essential tools for practitioners in selecting big data clustering algorithms;

- Using a modern metaheuristic optimization technique, Variable Neighborhood Search (VNS) [Bri+23], the proposed Big-means approach is further improved (Chapter 6). The modified algorithm is named BigVNSClust;

- The Big-means method is enhanced by incorporating an adaptive mechanism to stochastically optimize sample size through a parallel approach, once more establishing it as the new state of the art in the field of big data clustering. The new algorithm is named BigOptimaS3 (Chapter 7);

- We theoretically solidify the results and underlying concepts of this study by synthesizing them into a new metaheuristic, Variable Landscape Search (VLS). This approach offers a fresh perspective in optimizing search algorithms and possesses sufficient generality to encompass other comparable metaheuristic methods available in the literature (Chapter 8).

1.4 Dissertation outline

The remainder of this dissertation is systematically structured as follows. Chapter 2 initiates the discussion by introducing the fundamental and most critical developments in cluster analysis, essential for understanding the concepts presented in later chapters. Moving for-

ward, Chapter 3 offers a literature overview, establishes the criteria for "true big data" clustering algorithms, delves into the proposed Big-means method, and culminates with its experimental validation. In Chapter 4, the focus shifts to exploring the critical dimension of effective and efficient parallelization of the Big-means algorithm. This is followed by Chapter 5, which presents a detailed review of available K-means-like clustering techniques and other MSSC methods in big data contexts, supported by extensive experiments and an interpretation of these results through the LIMA dominance criterion [Bri+23]. Chapter 6 extends the Big-means method using metaheuristic optimization, while Chapter 7 introduces a new big data clustering algorithm that incorporates competitive stochastic sample size optimization, further refining the Big-means methodology. Chapter 8 synthesizes the insights from all previous chapters, framing them within the novel Variable Landscape Search (VLS) metaheuristic, which encapsulates the results obtained and opens new avenues for developing potent optimization heuristics. Finally, Chapter 9 concludes the dissertation and outlines promising future research directions.

For a detailed account of the experimental results, Appendices A, B, and C are dedicated to providing an in-depth examination of the findings.

Chapter 2

BACKGROUND

2.1 Chapter structure

Section 2.2 offers a foundational introduction to key optimization concepts, enabling the reader to thoroughly understand the content presented in the rest of the dissertation. Section 2.3 contains an overarching classification of existing clustering models and methods, essential for understanding various unavoidable trade-offs associated with their use. Section 2.4 traces the evolution of big data clustering technologies over time, offering readers a detailed timeline. Section 2.5 discusses the K-means algorithm in more detail and sets the stage for the subsequent chapters by marking specific opportunities for its optimization in the big data context. The two final sections introduce the reader to the essential concepts that underlie the design and evaluation of our algorithms proposed in the subsequent chapters. Section 2.6 is a basic introduction to the fundamental principle "the simpler, the better" in the context of optimization. Section 2.7 is devoted to the LIMA dominance criterion, which enables a more truthful comparison of different optimization algorithms in practice.

2.2 Elements of local and global optimization

A decision variable is an unknown quantity that directly affects the outcome of a given problem and is under control of a decision maker. Optimization is a mathematical procedure to select the best value of the decision variable from a set of available alternatives (possible solutions) with respect to some criterion [Hol14]. Often, the criterion is set to be the maximization or minimization of a real-valued function over the portion of its domain where some constraint relations are satisfied. This portion is called the feasible region, and the function in question is referred to as the objective function. Usually, at any given time the

optimization process only looks at a relatively narrower subset of the feasible region, which is called the set of all candidate solutions, or the search space.

Equipping the search space with some neighborhood structure, one can define a neighbor solution of a given solution to be a candidate solution contained in some neighborhood of the solution. The set of all neighbor solutions to a given solution is called the neighboring set of the solution.

Optimization can be of two main types: local and global. Global search is the process of finding the optimal value of a given function among all possible values of the decision variable in the feasible region. In contrast, local search starts from a candidate solution and then iteratively moves to neighbor solutions. At each step, a decision is made on which neighbor solution to move to based on maximization of some local criterion. Thus, local search algorithms are an example of greedy algorithms. The incumbent solution is defined to be the current best solution found during an algorithmic search procedure [Hol14]. When there are no improving neighbor solutions for the incumbent solution, then the local search is stuck at a locally optimal point. Although local search algorithms tend to be faster than global search algorithms, local search algorithms are generally not guaranteed to find the globally optimal solution, if one exists.

Termination of a local search can be triggered by either a time constraint or a limit on the maximum number of iterations without any improvement in the incumbent solution's quality. Local search is an anytime algorithm, capable of returning a valid solution even when interrupted at any point before completion. These algorithms typically exhibit improvement in solution quality over time, with the potential for better solutions the longer they run. Additionally, local search algorithms can be approximation algorithms, terminating before finding the optimal solution. These algorithms provide near-optimal solutions to optimization problems (often NP-hard) with proven bounds on the distance from the optimal solution. They strike a balance between solution quality and computational efficiency, making them a valuable tool for tackling complex problems. In particular, assume that the optimal solution C^* exists for the MSSC problem. Then, we say that an approximation

algorithm for the MSSC problem returns an A-approximate solution C, for some $A \geq 1$, if

$$f(C, X) \leq A \cdot f(C^*, X)$$

There are various methods to address the issue of local optima in local search algorithms. One approach involves utilizing multiple restarts with different initial conditions. Other strategies include more sophisticated iterative schemes like iterated local search [LOT10], memory-based techniques such as reactive search optimization [BB10], or memory-less stochastic modifications like simulated annealing [GMS19]. The latter approach is also referred to as the stochastic local search [HS05]. A pseudocode for this algorithm is described in Algorithm 1.

Algorithm 1: Stochastic Local Search

```
Function Stochastic_local_search(f):
    Input  : Function to be optimized f : X → ℝ;
    Output: Approximate solution x_c ∈ X.
    Generate initial solution x_c
    repeat
        Create new solution x = N(x_c)
        if f(x) ≤ f(x_c) then
            x_c ← x
    until convergence criteria is met;
    return x_c
```

Note that the function $N(\cdot)$ used in Algorithm 1 introduces stochastic perturbations into the choice of the next incumbent solution. This crucial step, known as "shaking", is introduced before each subsequent descent (local search) phase. The shaking step aims to introduce stochastic perturbations into the selection of the next incumbent solution. The goal is to apply a perturbation that is sufficiently strong to escape the current local optimum's basin of attraction. However, if the perturbation is too weak, the local search may frequently revert back to the previously visited local optimum, resulting in search stagnation. Conversely, if the perturbation is too strong, its effect can resemble a random restart of the

search process, which typically leads to a low probability of discovering improved solutions during the subsequent descent phase. Such a shaking step plays a vital role in the iterated local search [LOT10]. A pseudocode for this algorithm is shown in Algorithm 2.

Algorithm 2: Iterated Local Search

```
1  Function Iterated_local_search(f):
       Input  : Function to be optimized f : X → ℝ;
       Output: Approximate solution x_c ∈ X.
2      Generate initial solution x_0
3      x = Local_search(x_0)
4      x_c ← x
5      repeat
6          x' ← Shake(x)
7          x'' ← Local_search(x')
8          if f(x'') ≤ f(x_c) then
9              x_c ← x''
10         x ← Acceptance_criterion(x, x'', x_c)
11     until convergence criteria is met;
12     return x_c
```

The proposed Big-means algorithm (Chapter 3) is not an instance of the iterated local search, particularly for big data applications, as evaluating function f in step 8 of Algorithm 2 is computationally impractical in such contexts. Additionally, in the Big-means algorithm, the shaking procedure corresponds to sampling from an input dataset, while the local search procedure involves applying the K-means algorithm to a drawn sample rather than the entire dataset.

2.3 Classification of clustering algorithms

To provide the reader with a context of the available broad types of clustering algorithms, we present the following categories:

1. Partitioning-based algorithms. Partition-based clustering algorithms such as K-means [HJ82] or K-medoids [KR90b] divide data into non-overlapping subsets (clusters) such

that each data point belongs to exactly one subset. They typically start with an initial set of centroids and assign each data point to the nearest centroid to form clusters. These algorithms then iteratively update the centroids to minimize the total within-cluster variance or another similar measure. The main challenge with partition-based methods is the need to specify the number of clusters in advance. Also, they are sensitive to the initial choice of centroids and can get stuck in local optima, so multiple runs with different initial centroids may be necessary. Popular algorithms in this category include K-means [HJ82], K-medoids [KR90b], and CLARA [KR90a];

2. Grid-based algorithms. Grid-based algorithms, such as STING [WYM97] or CLIQUE [Agr+98], divide the data space into a finite number of cells to form a grid-like structure. The grid cells can be of various shapes and sizes, and the level of granularity of the grid is determined by a user-defined parameter. The clustering process is then performed on the grid structure by considering the cell densities, i.e., the number of data points in each cell. This allows the algorithm to be highly efficient since the complexity is tied to the number of cells in the grid, not the number of data points. However, the choice of grid size and shape can greatly influence the resulting clusters, and finding an appropriate grid configuration can be challenging;

3. Hierarchical algorithms. Hierarchical clustering algorithms are a type of clustering method that build a hierarchy of clusters either by a divisive method which starts with all observations in one cluster and divides them into smaller clusters, or an agglomerative method that starts with each observation as a separate cluster and merges them to create bigger clusters. Decisions on splitting or merging pairs of clusters are made based on some predefined metric on the set of all clusters. The result of this process is a tree-like diagram called a dendrogram, which shows the order of merges and the proximity at which they occurred, offering a comprehensive view of the data's hierarchical structure. Given a desired number of clusters, the corresponding clustering configuration can be efficiently retrieved by slicing the dendrogram at the appropri-

ate height. Some notable examples of algorithms in this category are Ward's [Jr63], BIRCH [ZRL96], and CURE [GRS98];

4. Density-based algorithms. Density-based algorithms are a category of clustering methods that identify clusters as regions of high data point density separated by areas of low density. These algorithms work by defining a neighborhood around each data point, and if a sufficient number of points is found within this neighborhood, a new cluster is created or an existing cluster is expanded. This approach allows the detection of arbitrary shaped clusters and has the ability to handle noise and outliers. Popular algorithms in this category include DBSCAN [Est+96], OPTICS [Ank+99], and DENCLUE [HK03];

5. Spectral clustering algorithms [Lux07; Hua+20]. Spectral clustering is a class of techniques that make use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. This method does not make strong assumptions about the form of the clusters, so it can capture complex structures quite well;

6. Model-based algorithms. Model-based clustering algorithms are a family of clustering algorithms that model each cluster as a sample from a probabilistic model. The goal is to identify the model that best fits your data. These algorithms assume that the data are generated by a mixture of underlying probability distributions, each corresponding to a different cluster. Popular algorithms in this category include the EM algorithm for Gaussian Mixture Models [YLL12] and Hierarchical Dirichlet Processes [Teh+06].

Among the above-mentioned classes, only the partitioning-based, notably K-means, excel in clustering large datasets due to their time complexity, without needing substantial modifications or the integration of additional approaches.

2.4 Timeline of main developments

The topic of devising parallel globally convergent scalable clustering algorithms has seen some sharp interest in the recent research works. In [LVV03] Likas et al. introduced the seminal idea of a global K-means algorithm (GKM). They introduced a novel deterministic incremental method. In each iteration, a new cluster center x is added, which represents the optimal solution from m runs of the K-means local search. This search is initialized at $\{c_1^{k-1}, \ldots, c_{k-1}^{k-1}, x\}$. Here, m stands for the total number of dataset entries, while c_i^j signifies the ith best cluster center selected during the jth iteration. The same study introduced an innovative heuristic termed the Fast Global K-means Algorithm (FGKM). In every step, this heuristic selects x to minimize an upper bound of the clustering error function.

It became clear that by selectively and greedily considering a smaller subset of candidates for improvement at each iteration, swifter heuristics could be developed. Pursuing this idea, Bagirov developed an approach to minimize the auxiliary clustering function, as detailed in [Bag08]. This minimization allowed for the selection of a single promising initial solution in each step, eliminating the need to assess all m options. This evolved algorithm, termed the Modified Global K-means Algorithm (MGKM), demonstrated superior accuracy compared to the GKM algorithm, albeit at a time cost. A more time-efficient variant, the Fast Modified Global K-means Algorithm (F-MGKM), was presented by Bagirov et al. in [BUW10]. Further enhancements were made in Ordin and Bagirov [OB15], culminating in the Multi-start Modified Global K-means (MS-MGKM) algorithm, which generates multiple candidate solutions at each step.

In their investigation of the MSSC problem, Bagirov et al. presented DCClust, an innovative incremental algorithm rooted in the DC representation, as documented in [BTU16]. Their findings highlighted that the DC formulation surpassed earlier incremental techniques, particularly for extremely large datasets. Additionally, the prospect of applying sophisticated non-smooth optimization methods to the non-smooth MSSC formulation seemed promising for further enhancements. Pursuing this line of thought, a novel large-scale clustering al-

gorithm named DCD-Bundle was introduced in [KBT17]. This algorithm draws inspiration from the diagonal bundle method, D-Bundle, detailed in [Kar15], tailored for sparse, large-scale, non-convex, and non-smooth optimization challenges. Comparative analyses revealed that DCD-Bundle matched DCClust's accuracy but exhibited superior computational speed.

Karmitsa et al. introduced LMBM-Clust in their work [KBT18]. This incremental technique is tailored for clustering large datasets, characterized by both a large number of attributes and data entries. At its core, LMBM-Clust merges the limited memory bundle method (LMBM) from [HMM07] as a local search strategy with the starting center generation method devised by Ordin et al. in [OB15]. During the experimental evaluation, LMBM-Clust method showed fairly good results both with respect to time and accuracy, while only losing in time to K-means++. In addition, LMBM-Clust always used less and in some cases significantly less distance function evaluations than other existing incremental approaches. Nevertheless, one of the major drawbacks of LMBM-Clust is its high complexity and a large number of hyperparameters. Using the original implementation of LMBM-Clust provided by the authors, this algorithm was unable to cluster a significant portion of large real-world datasets considered in our experimental evaluation due to the out-of-memory error.

I-k-means−+ was proposed by Ismkhan in [Ism18]. The primary idea of I-k-means−+ was to use various heuristics to iteratively improve the quality of the solution produced by K-means. This was achieved by removing one cluster (minus), dividing another cluster (plus), and reapplying clustering on the full dataset again. I-k-means−+ showed better results than K-means and K-means++ both on real and synthetic data, especially for an increasingly larger number of data points and clusters. Nonetheless, this algorithm also suffers from excessive complexity and usage of the whole dataset in every iteration.

In [GV19], Gribel et al. proposed HG-means. This is a hybrid approach based on the idea of multi-start K-means, where new initial centroids are selected using such genetic operations as mutation, crossover, and selection. The experiments showed that HG-means produced state-of-the-art results with respect to both quality and time, while being inferior in time

only to LMBM-Clust. However, in every iteration HG-means requires clustering of the full dataset, which renders the algorithm unsuitable for big data. Also, in this paper, the authors disputed the belief that a near-optimal solution of the MSSC has only a marginal impact on the clustering validity. They observed that this result becomes more pronounced as the dimension and the number of clusters increase. This critical observation justifies the belief that the key challenges associated to high-dimensional data clustering may be addressed by progressing toward faster and more accurate MSSC solvers, even before attempting to change the clustering model or paradigm.

In [CPL20], Capo et al. proposed a grid-based algorithm called the Boundary Weighted K-means (BWKM). The BWKM algorithm recursively applies a weighted version of the K-means algorithm on a sequence of spatial-based partitions of the dataset. These spatial-based partitions ideally contain a large amount of well-assigned blocks, i.e., cells of the spatial partition composed of instances with the same cluster affiliation. BWKM showed promising results in terms of accuracy over the Forgy K-means, K-means++, and Minibatch K-means approaches. Meanwhile, BWKM used significantly less distance function evaluations. Nevertheless, BWKM achieves its best results only for tall data, i.e., large datasets with small dimensions and relatively low number of clusters. Also, BWKM suffers from excessive complexity, which significantly hinders its efficient implementation in practice.

In [AAS21], Alguliyev et al. proposed a partitioning scheme that guarantees speedup for clustering very large datasets. This algorithm was named Big Data Clustering on a Single Machine (BDCSM). In the original paper, the conducted empirical evaluations of BDCSM concluded its superiority only over the classical K-means algorithm. In essence, the BDCSM algorithm slices the whole dataset into chunks, clusters the obtained chunks using K-means, aggregates the resulting cluster centers into a final pool, and produces final centroids as the result of clustering the compiled pool using K-means. A more detailed discussion of the BDCSM algorithm in a relevant context is postponed until the next Section 5.3.

2.5 K-means algorithm

As noted in the introduction, the K-means algorithm [HJ82] is a cornerstone of clustering methods, renowned for its efficiency and ability to deliver relatively high-quality local solutions in a computationally feasible time frame when applied to small or large data [Jai10]. K-means is regarded as one of the most popular methods for solving the MSSC problem [CYY20]. Its popularity stems from the method's high efficiency, coupled with its simplicity and versatility. A pseudocode for the K-means algorithm is provided in Algorithm 3.

Algorithm 3: K-means Clustering Algorithm

```
Function K_means(X, C):
    Input  : Feature vectors X = {x_1, ..., x_m};
             Maximum number of iterations T;
    Output: Final cluster centroids C = {c_1, ..., c_k};
             Point-to-cluster assignment A = {a_1, ..., a_m};
             Objective f(C, X).
    Initialize centroids C uniformly at random in the convex hull of X;
    Initialize the iteration counter t = 0;
    while there is a change in centroids C or t < T do
        for every point x in X do
            Assign x to its nearest centroid in C;
        end
        for each centroid c_i in C do
            Recalculate c_i to be the centroid (mean) of all points currently assigned
             to it;
        end
        Increment iteration counter t ← t + 1;
    end
    Allocate each point in X to its closest centroid in C, forming cluster assignments
     A;
    return C, A, f(C, X)
```

K-means is an iterative algorithm consisting of two alternating steps: assignment and update. The algorithm's input is an initial set of k centroids $C = (c_1, \ldots, c_k)$ and some

stopping criterion, which can be a maximum number of iterations or a tolerance on the distance between two consecutive solutions. In the assignment step, each point x_i is assigned to the nearest centroid from C. In the update step, the centroids are updated by assigning the means of points in each cell X_j of the resulting partition: $c_j = \frac{1}{|X_j|} \sum_{x_i \in X_j} x_i$. These two steps are repeated until the stopping criterion is met.

Due to its simple and straightforward structure, K-means is amenable to optimization at nearly every stage in general and specifically for big data. First, before running the K-means algorithm, it might be advisable to perform normalization or dimensionality reduction on the input data. Second, the choice of initial centroids (step 2 in Algorithm 3) is an important part of the K-means algorithm, which significantly impacts the accuracy of the resulting clustering solution [FS19]. The classical version of K-means algorithm initially assigns centroids to clusters using a uniform random initialization. However, it has been established that the clustering solution obtained with a random initialization of K-means may be significantly worse in terms of the objective function when compared to the optimal clustering solution. This means that relying solely on random initialization can lead to suboptimal clustering results [Hen+16]. Therefore, improving initialization is another method for optimizing the performance of K-means.

The initialization issue of the classical K-means algorithm can be addressed through two main approaches. The first one involves selecting the initial solution carefully, as suggested by Franti et el. [FS19] and Ismkhan et el. [Ism18]. The other approach is embedding the K-means algorithm into a global optimization method as a local search procedure, as proposed by Franti et al. [FS19]. Thus, numerous search methods, both local [HM01] and global [MS21], have been proposed to offer improved solutions.

One of the most effective heuristics for K-means initialization is K-means++ [AV07]. K-means++ works as follows: the first cluster center c_1 is chosen randomly from all data points. Each subsequent cluster center c_j is selected from the remaining data points with a probability proportional to its squared distance from the existing cluster centers. The pseudocode of K-means++ can be found in Algorithm 4. While K-means++ initialization takes longer

than a random initialization, it ensures that K-means requires fewer iterations to reach an optimum. Additionally, K-means++ often produces a smaller objective function value at the final solution compared to a naive random initialization. In [MRS20], Makarychev et al. demonstrated that the expected cost of the solution produced by K-means++ is at most $5(\log k + 2)$ times the optimal solution's cost. In the meantime, the time complexity of the K-means++ initialization is $\mathcal{O}(m \cdot n \cdot k)$. Hence, it is also not directly applicable to big data. However, the K-means++ algorithm can be efficiently parallelized for multi-core processing of large data. In K-means and K-means++, parallelization can be used to compute the distances between centers and data points. In K-means++, it is also possible to parallelize the computation of the probabilities for selecting each data point as a center.

Algorithm 4: K-means++ Initialization Method

1 **Function** `K_means_plus_plus(`X, k`):`
  Input : Feature vectors $X = \{x_1, \ldots, x_m\}$;
  Desired number of cluster centers k;
  Output: Cluster centers $C = \{c_1, \ldots, c_k\}$.
2 Choose the initial center c_1 uniformly at random from X;
3 **for** $i = 2$ **to** k **do**
4 Let $d(x_j)$ be the shortest distance from object $x_j \in X$ to the closest already chosen center $c_j \in \{c_1, \ldots, c_{i-1}\}$;
5 Choose the next center $c_i = x_t \in X$ with probability $\frac{d(x_t)^2}{\sum_{x_j \in X} d(x_j)^2}$;
6 **end**
7 **return** C

A simpler but less efficient initialization method is Forgy [For65], which also samples an initial solution from the original data points, but uniformly at random. Multi-start K-means [FS19] is yet another initialization method, which executes several random initializations and picks the best one in the end.

The main loop of the K-means algorithm (line 4 of Algorithm 3) can be parallelized both in a parallel multi-start manner and at the level of internal parallelization of steps within each iteration (distance calculations at step 6 of Algorithm 3). Also, the distance calculation

step at line 6 of Algorithm 3 can be optimized using the triangle inequality [Elk03; MS22].

During the operation of the K-means algorithm (line 5 of Algorithm 3), various "divide and conquer" techniques can also be applied, where initially the data are divided and distributed among parallel workers, solutions for small subtasks are obtained, which are then combined at the final stage to produce a solution for the entire task. Decomposition of input data into subtasks can be accomplished in multiple ways, such as simple uniformly random sampling [Mus+23] or more advanced techniques like coresets [BLK18] and canopies [MNU00]. The coreset method is known for its efficiency and accuracy in reducing the size of large datasets without losing important information.

During the parallelization process, advanced optimization methods can also be employed, such as collective or genetic algorithms, among others. In this approach, K-means is integrated as a local search procedure within higher-level heuristics such as MDEClust [MS21], HG-means [GV19], or Big-means [Mus+23]. K-means can be optimized by using High-Performance Computing (HPC) technologies such as vector computations based on SIMD technology and others implemented in Numba [LPS15; Mar18], among others.

2.6 "Less is more" approach

The "Less is More" Approach (LIMA) [MTU16] is a recent paradigm for solving optimization problems [Bri+17; Mla+19]. To solve a particular optimization problem, we must use or build the appropriate search algorithm. Having a complete list of possible search instructions, we can treat the process of building search algorithms as an optimization problem itself. In this sense, the "less is more" approach can be seen as a heuristic for this meta-optimization problem. More precisely, for the given optimization problem, there are two ways to optimize search algorithms:

1. Incorporating the minimum number of search instructions, which would make a search heuristic more efficient than the currently best in the literature (i.e., reducing the alphabet of possible base instructions without reducing the number of their executions);

2. Building a new heuristic that utilizes a smaller total number of executed search instructions, which would make the heuristic more efficient than the currently best in the literature (i.e., reducing the number of instruction executions without reducing their alphabet).

In other words, the main goal is to make a heuristic as simple as possible while keeping it more effective and efficient than the current state-of-the-art heuristic. The simplicity can lie in the heuristic structure or its behavior. Only reducing the instruction alphabet does not always lead to decreasing the overall number of the instruction executions in the search process. Obviously, the most "ideal" approach is to build such a search heuristic that simultaneously utilizes both possible ways, i.e., reduces the alphabet of base instructions as well as the overall number of their executions.

In this work, we will use both ways of the "less is more" approach for solving our optimization problem. We believe that discovering a simple and user-friendly heuristic for some particular problem, which at the same time is more effective and efficient than others, represents a contribution to science. This approach is in contrast to just combining methods without having good computational results, only to claim a new methodological contribution. This is often done in some hybrid approaches. By minimizing the number of ingredients in the search, the search space itself, or the number of iterations, it becomes much easier to answer the typical question, "Why is heuristic working well?" [Mla+19]. Building on the LIMA approach, we introduce Big-means, a novel heuristic for big data clustering that enhances the basic K-means algorithm, which significantly reduces data usage while achieving improved clustering accuracy and efficiency in terms of CPU time.

Embracing the "less is more" philosophy with K-means involves streamlining the search domain, reducing the number of assignment and update iterations, and minimizing the data analysis workload. From the LIMA's point of view, the K-means algorithm is one of the simplest and, at the same time, one of the most efficient clustering algorithms since it provides a relatively high-quality solution in a reasonably short period of time. Moreover, as will be

shown in Chapter 3, embedding the K-means algorithm into the proposed heuristic endows it with the missing global optimization properties. This eliminates the unwanted property of convergence only to a locally optimal solution, which is the main drawback of the naive K-means algorithm.

2.7 LIMA dominance criterion

As discussed above, the "less is more" approach (LIMA) is a principle that emphasizes the power of simplicity in algorithm design [Bri+23]. It asserts that simpler algorithms can often yield more effective results than their more complex counterparts. Despite its successful application in various fields such as genetic networks, medicine treatment, and neurosciences, among others, it is seldomly used for solving optimization problems.

LIMA fits well into search heuristic design since a heuristic is usually defined by several characteristics, including speed, accuracy, simplicity, flexibility, effectiveness, and robustness. However, there is no formal approach to consider these attributes. Therefore, a mathematical framework was developed to formally incorporate one of these attributes, simplicity, alongside the commonly used ones, speed (processing time) and accuracy (solution quality). This resulted in the LIMA approach, which provides a bridge connecting optimization to artificial intelligence and machine learning.

The usual way to compare different methods to solve an optimization problem is to assess the relative solution quality (accuracy) and algorithm efficiency (speed). The LIMA approach adds a comparison of the simplicity of each method to the mix.

Definition 2.7.1. *The LIMA number of an algorithm A used to solve an optimization problem P on a dataset D is the cardinality of the set of ingredients U used by A.*

Definition 2.7.2. *Algorithm B is LIMA-dominant over algorithm A in solving problem P on dataset D if it satisfies the following conditions:*

- *Accuracy: the objective function $f_B \leq f_A$;*

- *Speed: the running time $t_B \leq t_A$;*
- *Simplicity: the LIMA numbers satisfy $|U_B| \leq |U_A|$;*
- *At least one of the above conditions is satisfied strictly.*

The LIMA-dominance relation could be too strong if applied to each run and each test instance. One way forward is to introduce the concept of weak or average LIMA-dominance, i.e., after many runs on the same problem, we verify that average values satisfy $f_B \leq f_A$ and $t_B \leq t_A$.

The problem can be considered as a challenging three-objective optimization problem defined by accuracy, speed, and simplicity. There are several ways of tackling a multi-objective problem including weighted multi-objective lexicographic goal programming, among others. If one wants to stress attribute simplicity, the aim would be to solve the following problem:

$$\min_{U' \subseteq U} \{|U'| \mid f_B(k) \leq f^*(k), t_B(k) \leq t^*(k); k = 1, 2, \ldots, M\}, \tag{2.1}$$

where index k identifies instance $T^{(k)} \in T$; $T = \{T^{(1)}, T^{(2)}, \ldots, T^{(M)}\}$ is a bank of benchmark instances; $f^* = (f^*(1), f^*(2), \ldots, f^*(M))$ is a vector of objective function values obtained by state-of-the-art algorithm A on these test instances; $f_B = (f_B(1), f_B(2), \ldots, f_B(M))$ is a vector of objective function values obtained by algorithm B using only the ingredients in U'; and $t^* = (t^*(1), t^*(2), \ldots, t^*(M))$, $t_B = (t_B(1), t_B(2), \ldots, t_B(M))$ are, respectively, the running times of algorithm A and B.

The objective is to find subset U' with the minimum number of ingredients (i.e., the smallest LIMA number) that satisfies the stated conditions. If $|U'| < |U^*|$ (the number of ingredients used in A), then algorithm B dominates algorithm A in the LIMA sense. As noted above the "total" dominance implied may be replaced by a less stringent requirement such as "average" dominance or dominance by statistical hypothesis testing.

Chapter 3

HOW TO USE K-MEANS FOR BIG DATA CLUSTERING?

3.1 Chapter structure

The chapter is organized as follows. Section 3.2 provides a comprehensive review of existing methodologies in the field, categorizing principal algorithm groups and highlighting their distinctive features. Section 3.3 delineates the critical attributes and desirable qualities of "true big data" clustering algorithms, the development of which is the primary focus of our research. Section 3.4 offers a detailed exposition of our proposed algorithm. This is followed by Section 3.5, which delves into a thorough analysis of the algorithm. Section 3.6 details the experimental framework and discusses the results we have achieved. In Section 3.7, we validate and demonstrate the key aspects of our algorithm through tests on synthetically generated datasets. The chapter concludes with Section 3.8, where we summarize our findings and suggest directions for future research.

3.2 Related work

The existing clustering algorithms for MSSC can be classified into two main groups: traditional and alternative.

Traditional algorithms, widely recognized in the literature for their simplicity and effectiveness, have been the subject of extensive study. For solving the MSSC problem (1.1), the K-means algorithm has been widely accepted along with other algorithms based on it: Forgy [For65], K-means++ [AV07], multi-start K-means [FS19], and others. Ward's method [Jr63], also considered a traditional approach, exhibits performance comparable to recent advanced heuristics in terms of solution quality. Nevertheless, its requirement to compute the square distance matrix limits its applicability to small and medium-sized datasets.

Furthermore, the literature has seen a surge in more complex alternative algorithms, primarily designed to improve the solution quality and speed for the MSSC problem. The following methods are among the most advanced and recent alternative algorithms: MDEClust [MS21], HG-means [GV19], LMBM-Clust [KBT18], I-k-means-+ [Ism18], BWKM [CPL20], BDCSM [AAS21], Coresets [Moh+18]. The vast majority of the alternative algorithms are complex and hybrid by nature, i.e., they are often based on various meta-ideas or are combinations of several other simpler algorithms. Complex alternative algorithms often attain state-of-the-art in terms of solution quality, yet they may fall behind K-means in time efficiency by up to several orders of magnitude. In the meantime, there exists a deeply grounded belief that winning in quality has only a marginal impact on the clustering validity. Nevertheless, this belief is actively disputed in [GV19]. The widespread adoption of more complex hybrid approaches generally results in clustering solutions of higher quality. However, this trend also brings a concurrent increase in time complexity and challenges in comprehending the hybrid algorithm for its potential users. The growing complexity of alternative algorithms, not matched by increased speed, leads to distrust among potential users. This is due to the challenges in implementing and adapting these algorithms, as well as the presence of unexplored intricacies within them. These factors can cause unexpected problems in the future. Also, these problems explain why most alternative algorithms are not gaining wide acceptance in the literature and among potential users. Therefore, an important task is to develop newer, simpler algorithms that can effectively balance solution quality and running time. Currently, literature suggests that this equilibrium is best achieved through the K-means algorithm when initialized with K-means++ seeding.

Many alternative local [HM01] and global [MS21] search algorithms have been proposed to provide better solutions. However, no alternative algorithm has received such wide recognition and spread in the literature as K-means. The unpopularity of alternative algorithms can be explained by the excessive computational effort, high time costs, and their insignificant impact on the final clustering validity [GV19]. The combination of K-means and K-means++ remains the most standard approach in the literature for solving clustering problems.

Nevertheless, the classical scheme of using K-means together with simple initialization methods has two significant drawbacks [MS21]. First, the final clustering result largely depends on the initial centroids. Second, the quality of the obtained solutions can be much worse than the maximum possible, especially with an increase in data dimension and the number of clusters. The classical K-means algorithm can be improved by properly selecting the initial cluster centers or embedding it into some global optimization algorithm as a local search method [FS19].

The multi-start initialization method is most commonly employed to overcome the shortcomings of the classical approaches based on K-means. In its most straightforward implementation, K-means is run several times on the whole dataset, and the result yielding the best quality is selected in the end. In our proposed scheme, K-means++ is only used to initialize the first K-means execution on a sample drawn from the dataset. All subsequent launches are initialized with the best solution found among all the previous local search executions. In every K-means execution, only a drawn random sample is clustered instead of the entire dataset. Both of these simplifications significantly reduce the computational complexity of the multi-start approach. Using random samples instead of the entire dataset introduces the right amount of variability in intermediate solutions, which is a necessary component in the search for the globally optimal solution.

As a rule, K-means is used as a local search routine inside advanced heuristics, which follow the principle of customized multi-start local search [MS21; GV19; Ism18; KBD22]. The proposed algorithm is no exception to this rule. By embedding the K-means algorithm into advanced heuristics [MS21] as a local search routine, one can endow K-means with the missing global optimization properties. This approach would eliminate the unwanted property of convergence only to a locally optimal solution, which is the main drawback of the naive K-means algorithm [GV19]. According to the customized multi-start approach, a more advanced logic is employed to initialize separate local searches, which pays attention to the results of previous launches to perform a more optimal initialization for the current one. Various metaheuristics or meta-ideas are usually built into the multi-start process to

obtain optimal initializations. Some examples are genetic search [MS21; GV19], variable neighborhood search (VNS) [NMT17; HM01], Greedy Randomized Adaptive Search Procedure (GRASP) [KBD22], simulated annealing [SBB+19], tabu search [LCR+18], and others. Consequently, a topical task is to build such an algorithm that would allow organizing the global search process in big data conditions without using any known metaheuristics or meta-ideas. Thus, in the proposed algorithm, instead of metaheuristics, only the natural properties of the K-means local search are used to organize the global search for the optimal solution to the MSSC problem. In our algorithm, the multi-start approach is customized not only at the level of using a more advanced initialization logic but also at the level of choosing data for clustering.

Alternative algorithms are superior in quality to the standard ones. The LMBM-Clust algorithm can be considered as the state of the art (SOTA) among the alternative algorithms. Nowadays, LMBM-Clust provides the best ratio of quality and clustering speed. This property makes LMBM-Clust suitable for clustering large data. Although the BDCSM algorithm [AAS21] technically applies to big data since it does not require a complete pass through a dataset, our experiments [KMM20] showed that BDCSM does not provide a significant advantage in terms of the clustering quality in comparison with other algorithms. The vast majority of other classical and alternative algorithms do not apply to big data since they require either a few (coresets) or many (hundreds or thousands) full passes through the entire big data. Thus, for instance, HG-means spends more than 6 hours on clustering the Gisette dataset that consists of 13,500 objects and 5000 features with $k = 25$ [GV19]. The LMBM-Clust algorithm processes the same dataset within 1 hour. In the meantime, our Big-means algorithm clusters this dataset within 38 seconds, providing the best result in terms of quality. If we relax the quality requirements, our algorithm can complete the clustering of this dataset within an order of magnitude less time while only slightly decreasing in the resulting quality.

One of the necessary properties for applying a clustering algorithm to big data is its scalability to the size of a dataset [GV19]. For example, a clustering algorithm can achieve

scalability by producing results without using all the available objects in the dataset. Scalability can also be achieved via the decomposition (or partitioning) of the dataset into portions for their subsequent parallel processing [KMM20]. Suppose a clustering algorithm can relatively independently process each of the obtained data portions. In that case, it can be implemented on a distributed system, where each portion is processed on a separate node of the distributed computation system. Our work presents an algorithm that effectively integrates all of these scaling approaches.

3.3 "True big data" clustering algorithms

The analysis of the available literature shows that there exists a strong need for new "true big data" clustering algorithms in the computer science community. We define a "true big data" clustering algorithm to be the one possessing a combination of the following practically useful properties:

1. Attainment of the best balance between the simplicity of an algorithm, the quality of the clustering result, and the speed of convergence in comparison with other existing state-of-the-art algorithms;

2. Global search for the optimal solution in the conditions of big data without using any known global optimization metaheuristics;

3. Scalability, which is expressed in the following features:

   (a) Ability to configure the trade-off between the resulting quality and the speed of its attainment;

   (b) Equal effectiveness in clustering small, large, and big data;

   (c) Ability to improve the clustering quality with an increase in the number of processed objects from the dataset;

(d) Possibility of obtaining a result of acceptable quality from an incomplete use of all available objects in the dataset;

(e) Ability of parallel and distributed processing of a dataset, which is divided into portions;

(f) Ability of stream processing of newly incoming data portions.

For a newly developed algorithm, a combination of the above valuable properties will allow it to gain wide recognition in the literature, as well as be in demand among practitioners in pattern recognition. In this work, we attempt to create a new simple algorithm that would possess all the above properties.

We are proposing a novel algorithm that combines both the sampling and parallelization approaches in a unique way, maximizing not only efficiency, but the accuracy as well. In the subsequent sections, we will precisely define the new algorithm, consider its key properties, and provide an empirical comparison of the proposed algorithm with the competitive methods.

3.4 Proposed algorithm

3.4.1 Big-means algorithm

By using the problem decomposition approach for the MSSC problem, we have devised a new heuristic that enables faster and more precise clustering of big data. This heuristic has the flexibility to utilize any suitable MSSC algorithm as a local search procedure.

Big-means starts by randomly creating a sample S of size $s \ll m$ from the given dataset X, where s is much smaller than the total number of feature vectors m. The initial configuration of centroids, denoted as C, is obtained by applying the K-means++ algorithm to the first drawn sample. Then, the algorithm iteratively clusters each new incoming sample by the K-means algorithm initialized with the best set of centroids that have been obtained across all the already processed samples so far. In each iteration, such a set of the best centroids

Algorithm 5: Big-means Clustering

Function `Big_means(`X, k, s, T`):`
- **Input :** Feature vectors $X = \{x_1, \ldots, x_m\}$;
  Desired number of clusters k;
  Sample size s;
  Maximum time T;
- **Output:** Centroids $C = \{c_1, \ldots, c_k\}$;
  Point-to-cluster assignments $Y = \{y_1, \ldots, y_m\}$;
  Value of the objective function $f(C, X)$.

$C \leftarrow \{\varnothing_1, \ldots, \varnothing_k\}$;
$\hat{f} \leftarrow \infty$;
Set iteration counter $t = 0$;
while $t < T$ **do**
  Draw a random sample S of size s from X;
  for *each centroid* c *in* C **do**
    if c *is the empty set* **then**
      Reinitialize c using K-means++ on S;
    end
  end
  Compute new centroids C_{new} using K-means on S with initial centroids C;
  if $f(C_{new}, S) < \hat{f}$ **then**
    $C \leftarrow C_{\text{new}}$;
    $\hat{f} \leftarrow f(C_{\text{new}}, S)$;
  end
  $t \leftarrow t + 1$;
end
$Y \leftarrow$ Assign each point in X to nearest centroid in C;
return $C, \quad Y, \quad f(C, X)$

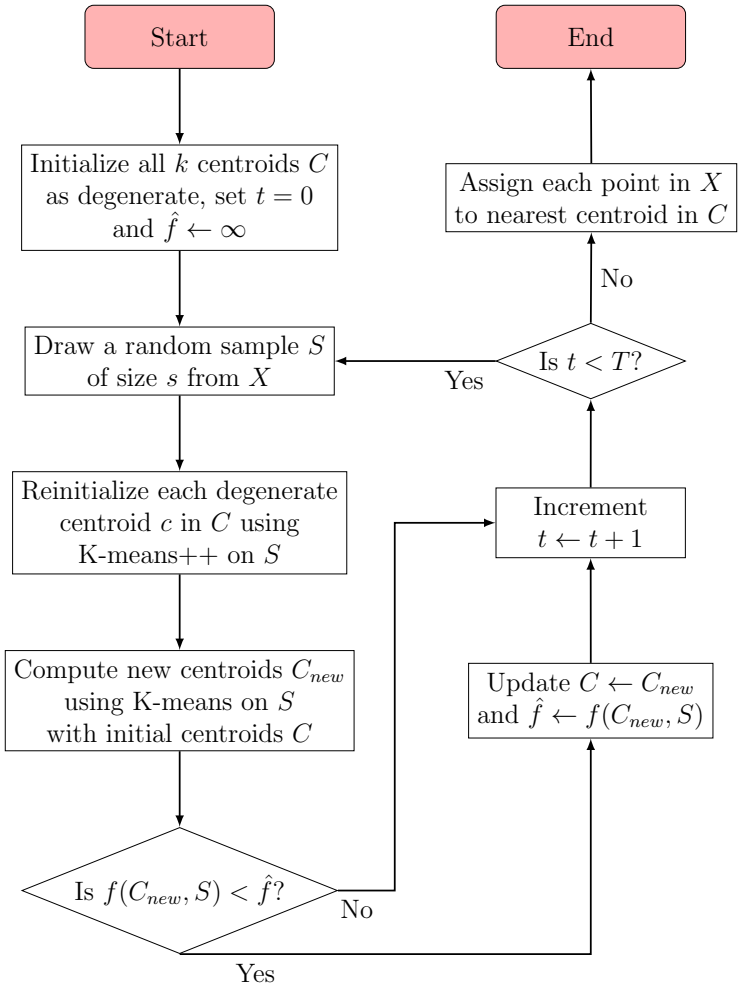


Figure 3.1: Flowchart of the Big-means algorithm

among the processed samples is called the incumbent solution. The criterion for choosing the incumbent solution is based on the objective function (1.1) evaluated on a sample. This iterative approach follows a "keep the best" principle, ensuring that the best solution found so far is always prioritized. Also, it is important to note that the current best (incumbent) solution C is replaced or kept intact in each iteration, without growing in size as the number of iterations increases.

Algorithm 5 shows the pseudocode of the proposed Big-means heuristic. Figure 3.1 shows a flowchart of Big-means. The source code of Big-means is available at https://github.com/R-

Mussabayev/bigmeans/.

In the Big-means algorithm, we used K-means as the local search heuristic and K-means++ as the initialization heuristic. However, Big-means can also accept other combinations of fast and accurate MSSC heuristics. Using K-means as the local search heuristic in Big-means is justified because K-means is one of the simplest and fastest local search heuristics among all MSSC algorithms. Also, the K-means algorithm has the following beneficial property: using better initial centroids (in terms of the objective function (1.1) minimization) results in faster convergence of the algorithm. Consequently, using better initial centroids on each subsequent sample leads to higher local convergence rates and speeds up the global search.

3.4.2 Initialization of degenerate solutions

Often, an initially non-empty cluster degenerates when all of its objects are redistributed between other clusters during the K-means iterations. At first glance, the effect of cluster degeneration may seem like a drawback. In reality, the emergence of degenerate clusters means that the clustering algorithm has managed to minimize the objective function using a smaller number of clusters. That is, the algorithm was able to pack the objects more densely into a fewer number of clusters. Hence, by adding the missing clusters using K-means++, we are almost guaranteed to obtain further improvements in the result.

In the Big-means algorithm, various methods can be employed to initialize the first sample [FS19], including a random initialization. Also, different strategies to deal with degenerate clusters can be used [Alo+17] instead of K-means++.

Big-means addresses degenerate clusters (also called empty clusters) differently than traditional approaches. When all data points initially associated with a cluster are reassigned to other clusters during the K-means process, Big-means reinitializes those empty clusters using the K-means++ algorithm. This introduces new potential cluster centers and enhances the overall clustering solution by providing more opportunities to minimize the objective function.

3.4.3 Final distribution of data points to clusters

Once a stop condition is met, such as reaching a limit on computational resources (e.g., the elapsed CPU time) or processing a maximum number of samples, the algorithm distributes the data points in the entire dataset into clusters based on their proximity to the final set of centroids C.

However, the resulting centroids C can also be considered a fully-fledged solution to the MSSC problem. Each fixed choice of centroids uniquely defines the distribution of data points across clusters. The last step of the Big-means algorithm simply redistributes the dataset points between the obtained centroids. Thus, the final step (19) of Algorithm 5 may not be necessary for many practical applications. For example, often it is enough to assign only a limited sample of objects to their closest centroids.

In fact, by omitting step 19, we can even consider Big-means as a new method for initializing K-means in big data conditions. More specifically, suppose that after step 19, K-means clusters the entire dataset starting from centroids C that resulted from Big-means. Although this may not be feasible for actual big data due to high computational costs, we should expect an even more significant improvement in the quality of the resulting clusters in terms of minimizing the objective function (1.1).

3.4.4 Sampling

In Big-means, it is crucial to have a way of obtaining data samples for their subsequent clustering. We have incorporated uniform sampling into Big-means. This kind of sampling does not calculate any joint probability distributions. It can be done with the total complexity of $\mathcal{O}(1)$, assuming that the sample size is a predefined constant number. Uniform sampling is the fastest possible sampling method and is optimal from the perspective of simplicity. Furthermore, under a mild assumption on the structure of dataset X, the authors of [HLD23] provide solid probabilistic guarantees for the simple uniform sampling applied to the MSSC clustering. Also, there is no need to randomly permute the feature vectors inside the dataset

since uniform sampling automatically breaks any orderings present.

The shaking procedure is a crucial element of the Big-means algorithm. It refers to creating new samples in each iteration, which perturbs the incumbent solution and introduces variability into the clustering results. This property is the next novelty of Big-means.

More precisely, in each iteration, we obtain a sparse representation of the original dataset by drawing a small, uniform random sample. If we represent the entire dataset as a point cloud in $\mathbb{R}^n$, then a sparse sample will approximate the spatial form of this cloud to some degree. Thus, decreasing the sample size increases the variability in the clustering results on the samples. The smaller the sample size, the stronger the incumbent solution is perturbed; vice versa, the bigger the sample size, the weaker the perturbation. Thus, by fine-tuning the sample size, one should find the suitable trade-off between the degree of variability and the degree of approximation of the original data form. An optimal choice for the sample size will lead to the desired optimal behavior of the algorithm. An increase in the variability between clustering results of different samples has a positive effect on the global search since each subsequent sample offers some changed cluster configuration, which is checked for optimality during the search process.

3.4.5 Parallel processing

Since the MSSC problem is NP-hard and applied to big data, it is important to develop MSSC algorithms that effectively use modern high-performance computing (HPC) technologies. HPC uses supercomputers and computer clusters to solve advanced computation problems. Since HPC networking clusters and grids use multiple processors and computers, scalability is an essential property of HPC algorithms. Scalability is achieved mainly due to the parallel processing of the analyzed data using many available computing nodes, computers, or processors. The inherent nature of the proposed Algorithm 5 conduces to its effective parallelization.

A comprehensive analysis of various strategies for parallelizing the Big-means algorithm is deferred until Chapter 4. In the experiments conducted in the present chapter, we have opted

for the straightforward approach of inner parallelism. According to the inner parallelism strategy, individual data samples are processed sequentially, while the clustering process is parallelized at the implementation level of the K-means and K-means++ methods. To be more specific, this entails that the primary loop of Big-means remains sequential, while the internal loops responsible for calculating the minimum distances from data points to centroids are parallelized within the K-means and K-means++ algorithms.

When using enormous datasets, questions may also arise about how to distribute the given dataset X among parallel nodes of a computing system. In our implementation of Big-means, all workers had access to the common dataset, and each worker formed a clustered random data sample for itself. However, other schemes can also partially distribute the dataset between the nodes. A more detailed study on the effectiveness of various Big-means parallelization and data distribution schemes is an interesting subject of future research.

3.5 Analysis of the algorithm

3.5.1 Main properties

In accordance with the MSSC decomposition approach (see Subsection 5.3.1 for more details), Big-means orchestrates the process of clustering big datasets in a manner that avoids the necessity of using the entire dataset. Instead, only a limited number of representative data samples are processed. The algorithm calculates and compares local objective values, which are the values of the objective function for subproblems, without the requirement to compute the global objective function value for the entire dataset at each iteration.

By utilizing the proposed algorithm with appropriately chosen parameters, one can achieve satisfactory clustering results by converging to a final set of centroids in fewer than one iteration over the entire big dataset. In some instances, an additional iteration may be necessary during the final stages of the algorithm to assign each data point to its nearest centroid; however, this step may be omitted in certain applications.

It is important to note that our algorithm yields substantial savings in RAM usage,

primarily attributable to its decomposition of the complete dataset into separate samples, which can subsequently be processed sequentially or in parallel.

The scalability of the proposed algorithm is easily adjustable by choosing the appropriate number of samples and their size, which are the main parameters of the algorithm. By adjusting these parameters, it is possible to find a balance between the amount of analyzed information and the accuracy of the obtained solution. The larger the analyzed dataset is, the more advantages our algorithm gives over other algorithms. This favorable property is in direct accordance with the concept that "the more data, the better".

3.5.2 Time complexity

It is well known that the time complexity of each iteration of the standard K-means algorithm is $\mathcal{O}(m{\cdot}n{\cdot}k)$ [HW79]. Usually, real big data may contain so many vectors that the convergence of the classic K-means algorithm and its different versions becomes unacceptably slow for most practical applications. A poor initialization of the K-means algorithm can lead to the exponential total number of iterations to attain convergence, as shown in [Vat11]. The time complexity of the K-means++ initialization is also $\mathcal{O}(m \cdot n \cdot k)$.

The time complexity of each iteration of Big-means is $\mathcal{O}(s \cdot n \cdot k)$, where s is the sample size. Since the convergence of Big-means can be attained even for a relatively small sample size, with $s \ll m$, the overall time complexity of Big-means can be much smaller than the time complexity of K-means and K-means++ applied to the entire dataset.

3.6 Experiments with real-world datasets

3.6.1 Algorithm selection

We conduct a performance evaluation of Big-means in comparison to other state-of-the-art MSSC algorithms designed to address the challenges posed by big data clustering. Following an extensive review of the existing literature, we encountered a notable absence of MSSC algorithms well-suited for the clustering of big datasets. An ideal algorithm for this task

should demonstrate scalability, meaning that the quality of the obtained solutions should improve with the increase in data volume. Additionally, its time complexity should not exhibit linear or superlinear growth relative to the dataset size.

In light of these requirements, we have curated a selection of state-of-the-art clustering algorithms for our experimental evaluations. These algorithms are especially suited for large datasets and can even handle big data, given sufficient computational resources. Our focus will be on the Forgy K-means, K-means‖ [B+12], and LMBM-Clust [KBT18] algorithms. From our prior experiments, in which we compared over 20 unique K-means initialization methods, we determined Ward's algorithm to be the most effective initialization method for K-means.

We decided to include Ward's and LMBM-Clust algorithms since they are more sophisticated, computationally consuming, and therefore more precise algorithms for the MSSC problem. In contrast, Forgy K-means, K-means++, and K-means‖ have been selected for their simplicity and efficiency, albeit at the expense of some precision compared to Ward's and LMBM-Clust algorithms.

3.6.2 Datasets

The performance of the selected algorithms was evaluated on 19 publicly available datasets. Descriptions of these datasets are provided in Table 3.1, with more details available on their corresponding webpages listed in Table 3.2. In addition to these, four datasets were normalized, bringing the total to 23 datasets.

All datasets comprised exclusively numeric features, without any missing values. They ranged from having as few as two attributes to as many as 5,000, and the number of instances varied from thousands (smallest being 7,797) to tens of millions (largest being 10,500,000). The datasets were deliberately unbalanced, with significant variance in their number of objects and features, as this allowed us to evaluate Big-means as a universal algorithm suitable for various dataset sizes. Thus, we evaluated the aggregate performance of Big-means on datasets of various sizes without prioritizing any of them. Furthermore, the use of the same

Table 3.1: Brief description of the datasets

Datasets	No. instances m	No. attributes n	Size $m \times n$	File size
CORD-19 Embeddings	599616	768	460505088	8.84 GB
HEPMASS	10500000	28	294000000	7.5 GB
US Census Data 1990	2458285	68	167163380	361 MB
Gisette	13500	5000	67500000	152.5 MB
Music Analysis	106574	518	55205332	951 MB
Protein Homology	145751	74	10785574	69.6 MB
MiniBooNE Particle Identification	130064	50	6503200	91.2 MB
MFCCs for Speech Emotion Recognition	85134	58	4937772	95.2 MB
ISOLET	7797	617	4810749	40.5 MB
Sensorless Drive Diagnosis	58509	48	2808432	25.6 MB
Online News Popularity	39644	58	2299352	24.3 MB
Gas Sensor Array Drift	13910	128	1780480	23.54 MB
3D Road Network	434874	3	1304622	20.7 MB
KEGG Metabolic Relation Network (Directed)	53413	20	1068260	7.34 MB
Skin Segmentation	245057	3	735171	3.4 MB
Shuttle Control	58000	9	522000	1.55 MB
EEG Eye State	14980	14	209720	1.7 MB
Pla85900	85900	2	171800	1.79 MB
D15112	15112	2	30224	247 kB

Table 3.2: Information about the used datasets

Datasets	URLs
CORD-19 Embeddings	https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge
HEPMASS	https://archive.ics.uci.edu/ml/datasets/HEPMASS
US Census Data 1990	https://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990)
Gisette	https://archive.ics.uci.edu/ml/datasets/Gisette
Music Analysis	https://archive.ics.uci.edu/ml/datasets/FMA%3A+A+Dataset+For+Music+Analysis
Protein Homology	https://www.kdd.org/kdd-cup/view/kdd-cup-2004/Data
MiniBooNE Particle Identification	https://archive.ics.uci.edu/ml/datasets/MiniBooNE+particle+identification
MFCCs for Speech Emotion Recognition	https://www.kaggle.com/cracc97/features
ISOLET	https://archive.ics.uci.edu/ml/datasets/isolet
Sensorless Drive Diagnosis	https://archive.ics.uci.edu/ml/datasets/dataset+for+sensorless+drive+diagnosis
Online News Popularity	https://archive.ics.uci.edu/ml/datasets/online+news+popularity
Gas Sensor Array Drift	https://archive.ics.uci.edu/ml/datasets/gas+sensor+array+drift+dataset
3D Road Network	https://archive.ics.uci.edu/ml/datasets/3D+Road+Network+(North+Jutland,+Denmark)
KEGG Metabolic Relation Network (Directed)	https://archive.ics.uci.edu/ml/datasets/KEGG+Metabolic+Relation+Network+(Directed)
Skin Segmentation	https://archive.ics.uci.edu/ml/datasets/skin+segmentation
Shuttle Control	https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)
EEG Eye State	https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State
Pla85900	http://softlib.rice.edu/pub/tsplib/tsp/pla85900.tsp.gz
D15112	https://github.com/mastqe/tsplib/blob/master/d15112.tsp

methodology and datasets as those in Karmitsa et al. [KBT18] enabled additional comparisons of our results. The twelve smallest datasets are identical to those used in [KBT18].

3.6.3 Experimental design and evaluation metrics

Each of the 23 datasets was clustered using each algorithm n_{exec} times into clusters of sizes: 2, 3, 5, 10, 15, 20, 25. Following the methodology of the experiments in [KBT18], some datasets ("Shuttle Control", "EEG Eye State" and their normalized versions) underwent additional clustering into 4 clusters. The total number of conducted experiments reached 29,464. The result of each experiment was analyzed for relative accuracy ε and spent CPU time t (measured in seconds). For each algorithm A, dataset choice X and number of clusters k, relative accuracy ε is defined as:

$$\varepsilon(\%) = \frac{100 \times (f - f^*)}{f^*}$$

where $f^* = f^*(X, k)$ is the best value of the objective function observed on the whole dataset X using k clusters. We rely on f^* values as cited in [KBT18] and on our own best-obtained values for larger datasets not analyzed in [KBT18], designated by asterisks. Throughout the experiments, we noted instances where certain algorithms attained accuracies surpassing the existing best-known benchmarks. Consequently, encountering negative values of relative accuracy ε in some cases is not unusual. In fact, this underscores the exceptional performance of an algorithm that has surpassed our initial expectations.

3.6.4 Implementation nuances and hyperparameter selection

The maximum CPU time of Big-means was capped at T seconds. For the Big-means algorithm, CPU time t is considered to be the time of the last change of the incumbent solution C. The CPU time t is measured in seconds.

For each dataset, a rule of thumb was used to determine the sample size s of Big-means. We adjusted s until neither increasing nor decreasing it improved the objective function

values.

The K-means clustering process on each sample was stopped when the number of iterations exceeded 300, or the relative tolerance between two consecutive objective function values was less than 10^{-4}. These parameter values were empirically chosen to achieve an optimal balance between performance and accuracy for Big-means. For K-means++, three candidate points were considered when generating the next centroid, choosing only the best one.

Paper [B+12] empirically proved that the choice of oversampling factor $l = 2 \cdot k$ and number of rounds $r = 5$ in K-means‖ is optimal for most datasets, where k is the desired number of clusters. Thus, this value of l was used in K-means‖ for all datasets, whereas $r = 5$ was used for the largest six datasets and $r = \log(f(X, \{c_1\}))$ for the rest, where c_1 is the first randomly chosen centroid.

3.6.5 Software and hardware

For optimal comparability between the algorithms, we conducted all experiments on a consistent computer platform, utilizing the same programming language and versions of numeric libraries throughout. Specifically, our experimental setup employed Python 3.6, the NumPy 1.21 library for matrix and array computations, and Numba 0.46.0 for just-in-time compilation, which translates Python code into optimized machine code.

The experiments were conducted on a computing environment running Ubuntu Linux 4.4, featuring an x86_64 architecture with 8 CPU cores powered by Intel(R) Xeon(R) Silver 4114 processors, each clocked at 2.20GHz. Additionally, our system was equipped with 504 GB of RAM. This uniform hardware and software configuration ensured a level playing field for assessing the algorithms' performance.

The parallel implementations of K-means and K-means++ utilized all 8 CPU cores according to the first parallelization scheme described in Section 4.4. The same parallel versions of K-means and K-means++ were used inside Forgy K-means and Big-means algorithms. Thus, Big-means processes data samples sequentially, but each data sample is clustered us-

ing the parallelized versions of K-means and K-means++. The K-means‖ algorithm was parallelized with respect to the number of data points according to scheme one described in Section 4.4. We used the implementation of Ward's algorithm from the hierarchical clustering module of the SciPy library. The implementation of the LMBM-Clust algorithm was copied from the repository provided by the authors in [KBT18] and wrapped for Python through the Cython library.

3.6.6 Summarizing scores

For each experiment (choice of a dataset, algorithm, and value of k), we calculated the minimum, mean, and maximum values of the relative accuracy ε and CPU time t based on n_{exec} executions of the experiment. Appendix A provides detailed information on all experiments conducted. In the tables of Appendix A showing clustering details, n_d denotes the total number of distance function evaluations performed by an algorithm across different executions and all choices of k, while n_s stands for the total number of processed samples by the Big-means algorithm.

A summary of the performance of Big-means across all datasets is shown in Table 3.3. Performance is measured according to the following score system. Score S for algorithm A and performance metric q on dataset X is calculated as

$$S(A, X, q) = 1 - \frac{q_X(A) - \min_{A'} q_X(A')}{\max_{A'} q_X(A') - \min_{A'} q_X(A')}$$

where $q_X(A)$ stands for the value of performance metric q on dataset X resulting from algorithm A. In our experiments, we used the two main performance metrics: accuracy ε and CPU time t. To summarize the performance of a given algorithm across all dataset, we also introduce the sum score:

$$S(A, q) = \sum_{X \in \mathbb{X}} S(A, X, q)$$

where $\mathbb{X} = \{X_1, \ldots, X_{23}\}$ stands for the set of all used datasets X. The mean score can be

calculated as follows:

$$M(A, X) = \frac{1}{2} \sum_{q \in \{E_A,\ cpu\}} S(A, X, q)$$

Finally, the sum mean score is defined as

$$M(A) = \sum_{X \in \mathbb{X}} M(A, X)$$

Score $S(A, X, q)$ is in the range from 0 to 1 and can be interpreted as a normalized measure of the performance of a given algorithm A relative to the chosen performance metric q and dataset X in comparison with other algorithms A' that resulted on this dataset. The value of 1 indicates the best result among all algorithms, whereas the value of 0 indicates the worst result.

We summarize the performance scores of all algorithms in Table 3.4. Since the score $S(A, X, q)$ for each dataset X does not exceed 1, the maximum sum score $S(A, q)$ for every algorithm A and metric q is 23 points. Table 3.4 contains the percentage scores of all algorithms relative to this maximum and the sum scores restricted only to the first half of $\mathbb{X}$, which contains the biggest datasets.

In Appendix A, the detailed results of the experiments are shown in Tables A.1 – A.46 and Figures A.1a – A.2k. Some algorithms failed due to requiring too much RAM or time, and we filled their results with the "−" sign. These results were also not considered in the computation of final scores. In other words, if some algorithm fails in an experiment because of the "out of memory" error, this algorithm is given no scores for the experiment. The plots of the distance function evaluations for the Ward's and LMBMClust methods were not included in the figures since their results are several orders of magnitude larger than those of other competitive algorithms. However, one can easily retrieve these numbers from the corresponding numerical tables, if necessary.

Table 3.3: Summary of the performance of Big-means in accuracy ε and CPU time t over all datasets

Datasets $X_i \in \mathbb{X},\ i = 1, \ldots, 23$	$S(\text{Big-means}, X_i, q)$	
	$q = \varepsilon$	$q = t$
CORD-19 Embeddings	0.997	1.000
HEPMASS	0.999	1.000
US Census Data 1990	0.964	1.000
Gisette	1.000	0.980
Music Analysis	0.919	1.000
Protein Homology	1.000	0.993
MiniBooNE Particle Identification	1.000	0.988
MiniBooNE Particle Identification (normalized)	1.000	1.000
MFCCs for Speech Emotion Recognition	0.973	1.000
ISOLET	0.902	0.964
Sensorless Drive Diagnosis	0.978	0.998
Sensorless Drive Diagnosis (normalized)	0.936	1.000
Online News Popularity	0.993	1.000
Gas Sensor Array Drift	0.908	0.947
3D Road Network	0.971	1.000
Skin Segmentation	1.000	1.000
KEGG Metabolic Relation Network (Directed)	0.970	0.998
Shuttle Control	0.965	0.998
Shuttle Control (normalized)	0.892	1.000
EEG Eye State	1.000	0.826
EEG Eye State (normalized)	0.990	0.946
Pla85900	1.000	0.999
D15112	0.867	0.883
Sum score:	22.222	22.519
Maximum possible sum:	23.0	23.0
Performance score percentage:	$22.222/23.0 \times 100\% = 97\%$	$22.519/23.0 \times 100\% = 98\%$
Sum score for the first half:	10.731	10.924
Maximum possible sum for the first half:	11.0	11.0
Performance score percentage for the first half:	$10.731/11.0 \times 100\% = 98\%$	$10.924/11.0 \times 100\% = 99\%$

Table 3.4: Summary of the sum scores of all algorithms

Algorithm	$S(A, \varepsilon)$	$S(A, t)$	$S(A, \varepsilon), \%$	$S_{first\ half}(A, \varepsilon), \%$	$S(A, t), \%$	$S_{first\ half}(A, t), \%$	$M(A), \%$
Big-means	**22.222**	22.519	**97**	**98**	**98**	**99**	**97**
Forgy K-means	13.642	**22.632**	59	72	**98**	97	79
Ward's	11.936	1.574	52	18	7	7	29
K-means++	22.038	20.83	96	97	91	81	93
K-means‖	0.699	21.019	3	0	91	88	47
LMBM-Clust	19.944	7.981	87	81	35	8	61

3.6.7 Discussion of results

Big-means earned 22.222 points for accuracy and 22.519 points for time. This result corresponds to the percentage of 97% with respect to accuracy and 98% with respect to CPU time.

We sorted the datasets in Table 3.1 in the descending order of their size. If we evaluate the performance of Big-means only for the first half of Table 3.1, which contains the largest datasets, then the results are 98% in accuracy and 99% in time. Therefore, when the size of a dataset increases, the superiority of our algorithm also increases both in accuracy and time. Generally, our algorithm is more likely than others to show the best or comparable results in terms of accuracy for all datasets. Furthermore, Figures A.1a – A.1f show that Big-means performed significantly less distance function evaluations than other algorithms on the largest datasets. As the dataset size grows, the benefits of our algorithm become more pronounced. This property of Big-means is most useful for big data clustering. On the other hand, for all but the smallest datasets, Forgy K-means and K-means++ exhibited the largest number of distance function evaluations.

For the four smallest datasets, Big-means was still able to achieve the resulting accuracy that is better or comparable to the accuracies achieved by K-means++ and LMBM-Clust. Nevertheless, this was at the cost of more distance evaluations, as seen from Figures A.2e – A.2k. However, the increased number of distance evaluations did not significantly impact the resulting CPU time since Big-means still finished the clustering for these datasets within fractions of a second. Also, we observe that normalization favorably impacted the performance of Big-means for some datasets, such as MiniBooNE Particle Identification, Sensorless Drive Diagnosis, Shuttle Control, and EEG Eye State. After normalization, Big-means outperformed the competitive algorithms on these datasets both in accuracy and time. Furthermore, normalization allowed using much smaller sample sizes s for Big-means and significantly reduced the number of distance function evaluations. In the future research, we are planning to study the reasons for this behavior by testing Big-means on specifically

crafted synthetic data.

Thus, experimental results confirmed that the proposed algorithm fully satisfies all the requirements for a "true big data" algorithm formulated in Section 3.3. In particular, Big-means managed to complete the clustering of all datasets. This result is unlike other competitive algorithms since some of them failed to process the largest datasets due to the "out of memory" error.

3.7 Experiments with synthetic datasets

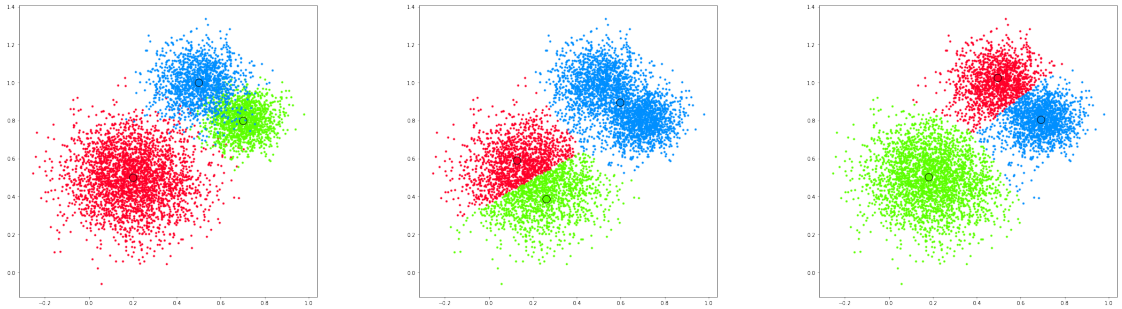
3.7.1 Experiments

In every Big-means iteration, the sampling process sparsifies the given dataset to some extent. We anticipate that this sparsification enables the initial centroids to shift more freely between the clusters, provided the clusters overlap. In essence, sampling with a carefully selected sample size enhances the flexibility of centroids within the current solution while still maintaining a reasonable approximation of the original dataset. Consequently, it is crucial to strike the appropriate balance between these two objectives when determining the sample size.

To examine this hypothesis further, we have conducted additional experiments wherein we assess the performance of Big-means using carefully generated synthetic data.

First, consider a mixture of 3 isotropic Gaussian distributions $\{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3\}$ with standard deviations $0.15, 0.08, 0.1$, respectively. Let $\mu_1 = [0.2, 0.5]^T, \mu_2 = [0.7, 0.8]^T, \mu_3 = [0.5, 1.0]^T$ be the coordinates of the means of these distributions. Let X_1 be the dataset obtained by sampling 3000 points from $\mathcal{N}_1$, 1500 points from $\mathcal{N}_2$, and 1500 points from $\mathcal{N}_3$. Dataset X_1 is shown in Figure 3.2a. If we assign the initial centroids $C = \{c_1, c_2, c_3\}$ as $c_1 = [0.1, 0.2]^T, c_2 = [0.1, 0.15]^T, c_3 = [0.5, 1.0]^T$, then the result of running K-means++ on X_1 with C as initial centroids is shown in Figure 3.2b. The value of the objective function $f(\{\mu_1, \mu_2, \mu_3\}, X_1)$ for the ground truth is 171.5. We see that K-means++ was trapped in a local minimum with the objective value of 191.52. Choosing the sample size $s = 70$ and $T = 1.5$, it is possible

to obtain a result of Big-means as shown in Figure 3.2c, with the objective value equal to 172.08. Hence, Big-means managed to avoid being trapped in a bad local minimum.



(a) (b) (c)

Figure 3.2: In 3.2a: the original dataset X_1; in 3.2b: the result of running K-means++ on X_1 with C as initial centroids; in 3.2c: the result of running Big-means on X_1 with C as initial centroids

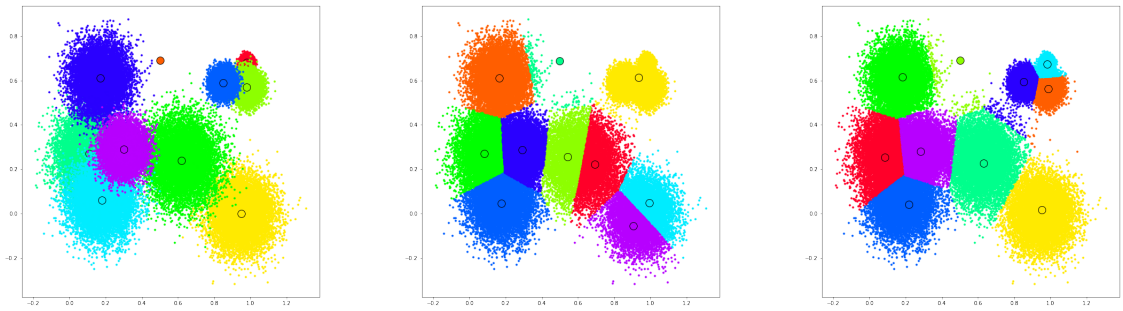
To consider a more involved example, we create another mixture of 10 isotropic Gaussian distributions $\{\mathcal{N}_1, \ldots, \mathcal{N}_{10}\}$ with the means

$$\{(0.98, 0.68), (0.50, 0.69), (0.95, 0.00), (0.98, 0.57), (0.62, 0.24),$$
$$(0.11, 0.27), (0.18, 0.06), (0.85, 0.59), (0.17, 0.61), (0.30, 0.29)\}$$

and standard deviations $\{0.015, 0.002, 0.084, 0.036, 0.097, 0.085, 0.085, 0.031, 0.079, 0.058\}$, respectively. Let X_2 be the dataset obtained by drawing a total of $100,000$ points from this mixture with equal weights for each component. The value of the objective function for the ground truth is 756.23. The results of running K-means++ and Big-means on X_2 are shown in Figures 3.3b and 3.3c, respectively. K-means++ was trapped in a local minimum with the resulting objective value of 843.02, whereas Big-means obtained the result of 772.47 with the sample size $s = 350$ and $T = 3.0$.

Thus, the experiments with datasets X_1 and X_2 allowed us to verify that the natural shaking property of Big-means helps it avoid falling into bad local minima.

Now, consider a mixture of 4 isotropic Gaussian distributions $\{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4\}$ with



(a) (b) (c)

Figure 3.3: In 3.3a: the original dataset X_2; in 3.3b: the result of running K-means++ on X_2; in 3.3c: the result of running Big-means on X_2

standard deviations 0.08 and means $\mu_1 = [0.2, 0.2]^T, \mu_2 = [0.3, 0.6]^T, \mu_3 = [0.8, 0.3]^T, \mu_4 = [0.6, 0.5]^T$, respectively. Let X_3 be the dataset obtained by sampling 100 points from $\mathcal{N}_1$, 100 points from $\mathcal{N}_2$, 3000 points from $\mathcal{N}_3$, and 900 points from $\mathcal{N}_4$. Dataset X_3 is shown in Figure 3.4a. The results of running Big-means on X_3 with $T = 2$ and sample sizes $s = 50, 200, 500$ are shown in Figures 3.4b-3.4d. The value of the objective function for the ground truth of dataset X_3 is 48.52. The resulting objective values for the 3 consecutive runs of Big-means are 58.83, 55.13, and 43.98, respectively.

The experiments with dataset X_3 showed that in the case of unbalanced clusters Big-means might require larger sample sizes in order to work efficiently. Since Big-means draws uniform samples, the probability of sampling some points from each cluster is proportional to the number of points this cluster has. This way, each cluster imposes its own threshold on the sample size, below which the contribution of the sampled points from that cluster becomes negligible and is not enough for the local search to detect it as a separate cluster. Therefore, the most efficient sample size should be a function of the size of the smallest cluster present in the dataset.

To see how Big-means performs on overlapping and non-overlapping (well-separated) balanced data, consider datasets X_4 and X_5 that are obtained by sampling $5,000,000$ points from 24 equally spaced Gaussian blobs on the regular grid $[0, 1]^2$ with standard deviations

0.01 and 0.025, respectively. Datasets X_4 and X_5 are depicted in Figures 3.5a and 3.5d. The results of running K-means++ and Big-means on datasets X_4 and X_5 are shown in Figures 3.5b, 3.5e, 3.5c, and 3.5f. Notice that in spite of a large number of entries in the dataset, a sample size of 2000 was enough for Big-means to very quickly obtain resulting accuracies than are comparable or better to those obtained by K-means++ on the entire datasets.

Finally, we investigate how the quality of the resulting solution and the clustering time of Big-means depend on various characteristics of the clustered data: the overall number of points, the dimension of the dataset, and the number of clusters. In these set of experiments, Big-means is compared to its closest rival, K-means++.

In our first scenario, we generated datasets with 10 Gaussian clusters, featuring random standard deviations between 0 and 10, 10 features, means within the range of $(-40, 40)^{10}$, and a sample size determined by the formula $m = 3^{i+7}$, where $i = 0, \ldots, 7$. To each synthetic dataset, we added 1000 points of uniform noise within the box $(-50, 50)$. For Big-means, we set $s = \min\{5000, m - 1000\}$ and $T = 3.0$. For each sample size m, we created a new dataset and ran each algorithm five times. The median results are displayed in Figure 3.6. The plots show that Big-means consistently delivers high-quality solutions, outperforming K-means++ across most sample sizes (m). Notably, Big-means exhibits superior stability and produces robust results, with computation times bounded below 3 seconds. This is a significant advantage for big data processing, where efficiency and reliability are crucial.

In the second scenario, we considered datasets containing 3^i Gaussian clusters for $i = 0, \ldots, 7$ with uniformly random standard deviations in the range $(0, 10)$, 10 features, uniformly random means in the region $(-40, 40)^{10}$, and 500000 points. To each synthetic dataset, we added 1000 points of uniform noise within the box $(-50, 50)$. The parameters $s = 10000$ and $T = 3.0$ were chosen for Big-means. The median results are shown in Figure 3.7. Again, the plots demonstrate that Big-means achieves better solution quality than K-means++ for most choices of m, only yielding to K-means++ at the last choice of m. Notably, K-means++ exhibits a drastic monotonic increase in processing time, while Big-means shows robust time results, consistently confined within a small constant range of 3 seconds. The results suggest

that for a constant sample size, Big-means exhibits a monotonically deteriorating curve of clustering accuracy. However, we expect that adaptively increasing the sample size with the increase in the number of clusters should resolve this issue.

In the last case, we synthesized datasets containing 10 Gaussian clusters with uniformly random standard deviations in the range $(0, 10)$ and uniformly random means in the region $(-40, 40)^{2^i}$, 2^i features for $i = 0, \ldots, 7$, and 500000 points. The noise of 1000 points was added, drawn uniformly in the box $(-50, 50)$. The parameters $s = 10000$ and $T = 3.0$ were chosen for Big-means. The median results are presented in Figure 3.8. Again, Big-means demonstrates robustness and high efficiency with respect to time. Moreover, Big-means delivers stable and accurate clustering results, unaffected by the number of features, whereas K-means++ produces either unstable and fluctuating or monotonically increasing results with respect to both metrics.

3.7.2 Summary

The experiments with synthetic data have concluded with the following key observations:

1. Big-means satisfies the claimed approximation property. Specifically, even when using sample sizes significantly smaller than the full dataset, Big-means achieves a solution quality that either matches or exceeds that of K-means++ applied to the complete dataset. Remarkably, if cluster sizes are fairly uniform, Big-means can either outperform or closely approximate the performance of standard K-means++ with just a minimal sample size;

2. With the right sample size selection, Big-means can circumvent bad local minima, which regular K-means++ inevitably falls into. This property holds only if there is an overlap between clusters;

3. Big-means performs similarly to K-means++ on the entire dataset when dealing with well-separated clusters. Specifically, for such clusters, if multiple initial centroids land

within a single cluster, proper redistribution becomes unattainable. Consequently, the local search is highly likely to settle into an undesirable local minimum. However, this problem can be considerably alleviated by introducing an additional shaking of the clustering solution that reinitializes an appropriate portion of the incumbent centroids in each iteration. An algorithm implementing this technique, BigVNSClust, will be proposed and shown to surpass the original Big-means algorithm in Chapter 6;

4. For datasets that are unbalanced, where there is a notable discrepancy in the number of points across clusters, Big-means may necessitate larger sample sizes. These sizes are particularly influenced by the smallest cluster in the dataset. Consequently, exploiting the advantageous shaking property of Big-means can be challenging in these scenarios. The primary reason is the difficulty in uniformly sparsifying all clusters while working with reduced sample sizes. Nevertheless, as will be shown in Chapter 7, augmenting Big-means with an adaptive automatic technique of adjusting the sample size tackles this problem effectively and efficiently. The resulting advanced algorithm is named BigOptimaS3.

3.8 Conclusion and future research

This chapter introduces a novel and highly scalable parallel clustering algorithm tailored for the MSSC problem. Through both analytic and empirical comparative studies, we pitted our proposed Big-means algorithm against other leading MSSC algorithms. Impressively, Big-means stood its ground, delivering comparable or even superior results for smaller datasets. Moreover, it consistently surpassed other algorithms in both accuracy and efficiency when dealing with large and big data. This contradicts the prevailing notion that clustering only a subset of data compromises accuracy compared to processing the entire dataset [MHE21a]. The elegance in Big-means' design suggests that resorting to intricate hybrid methodologies is not a prerequisite for high-quality clustering solutions. We seamlessly integrated global search and decomposition into our algorithm, bypassing traditional metaheuristics. A deeper

dive into big data concepts underscores the necessity of using decomposition to effectively tackle big data clustering challenges.

Speed, a small number of parameters, and the significant role of intuition in their choice are the prominent characteristics of Big-means' that are highly sought-after among practitioners in the field of pattern recognition. The utility of Big-means extends across a diverse range of applied fields where both big data and pattern recognition are pivotal. Due to the sample-based processing, Big-means also addresses the velocity aspect of the big data paradigm, which has rarely been addressed before. Another main advantage of the proposed algorithm is its global optimization nature when applied to big data. With few exceptions, most previously proposed big data clustering algorithms were local search algorithms.

However, some apparent shortcomings of the proposed approach should be addressed in future studies. These drawbacks include the need for precise analytic formulas or adaptive automatic techniques to determine the optimal sample size and runtime (addressed in Chapter 7), the inability of random sampling to handle highly imbalanced or well-separated clusters (addressed in Chapter 6), and the absence of formal mathematical proofs of the convergence properties of Big-means.

Future research on Big-means might encompass the following goals:

1. Optimizing the parallelization approach of Big-means (Chapter 4);

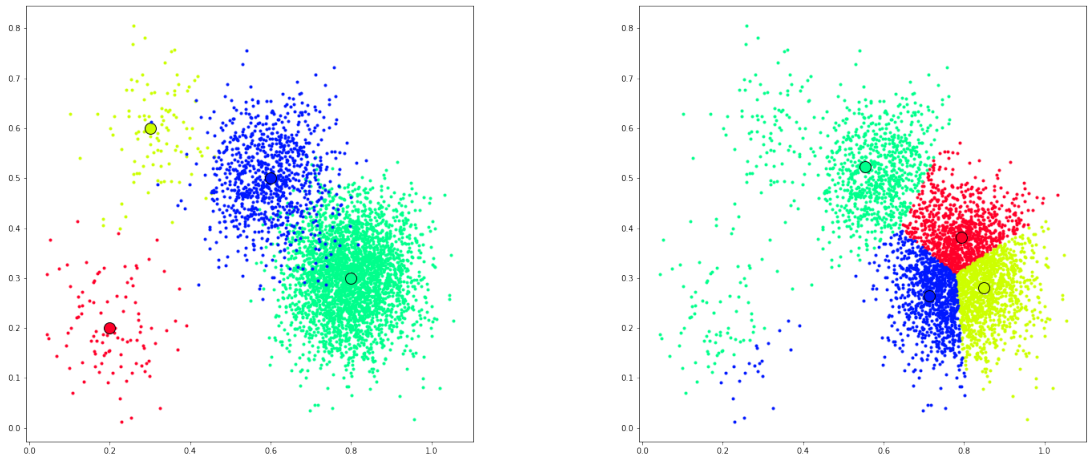
2. Generalizing the main principle of Big-means, which alters input data to perturb the objective function landscape, to a novel metaheuristic (Chapter 8);

3. Extension of Big-means to unstructured, heterogeneous, and high-dimensional data;

4. Providing a comprehensive theoretical framework for Big-means, including a convergence proof, approximation constant estimation, and convergence rate analysis.

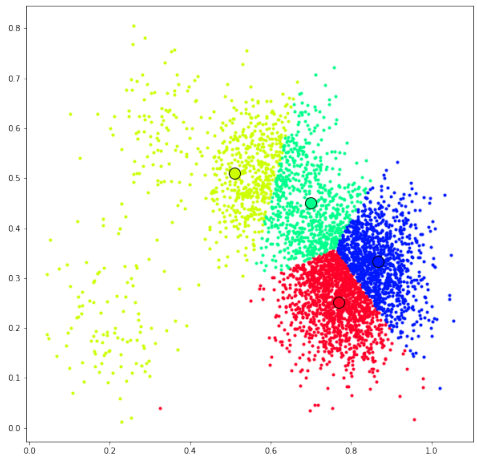
These results would allow us to better understand the properties of Big-means, increase its performance, and further extend it to address the variety issue of the big data paradigm. This

future work will significantly increase the utility of Big-means for practitioners in pattern recognition.

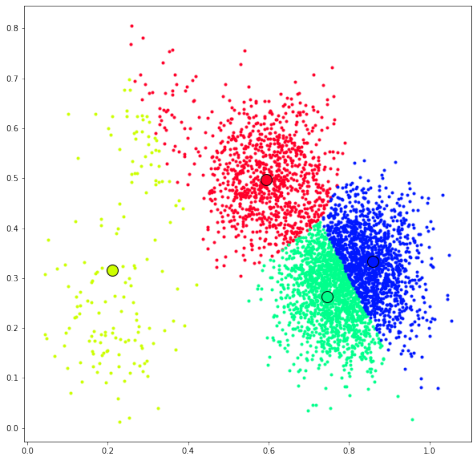


(a)

(b)

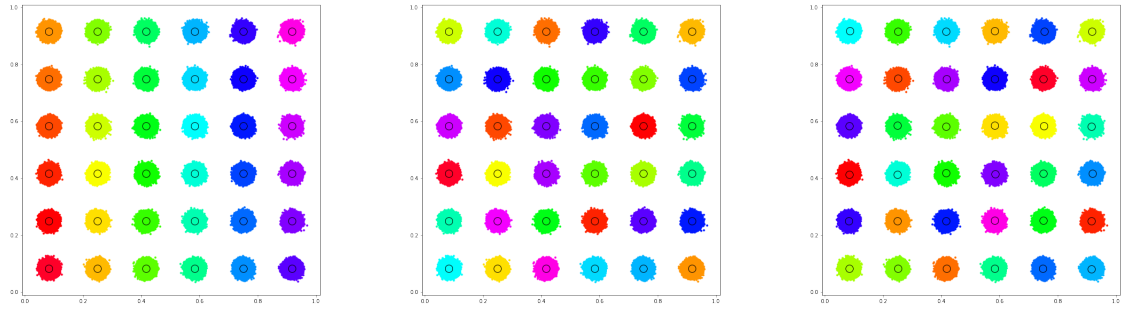


(c)

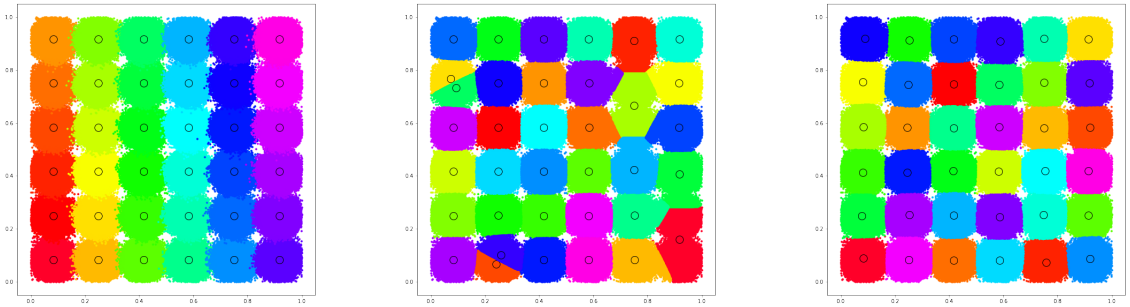


(d)

Figure 3.4: In 3.4a: the original dataset X_3; in 3.4b, 3.4c, 3.4d: the results of running Big-means on X_3 with the sample sizes $s = 50, 100, 200$, respectively

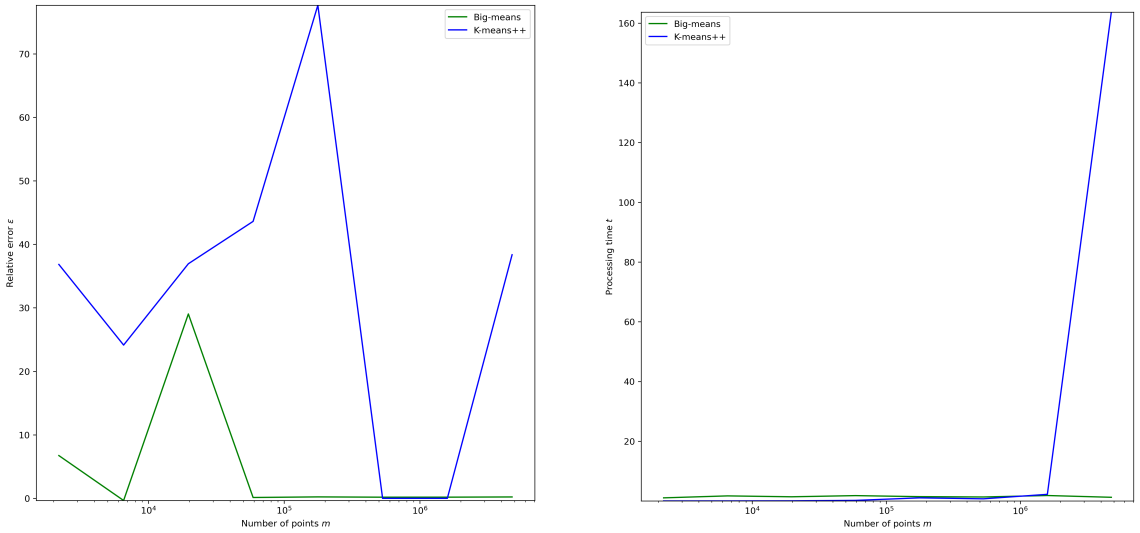


(a) (b) (c)



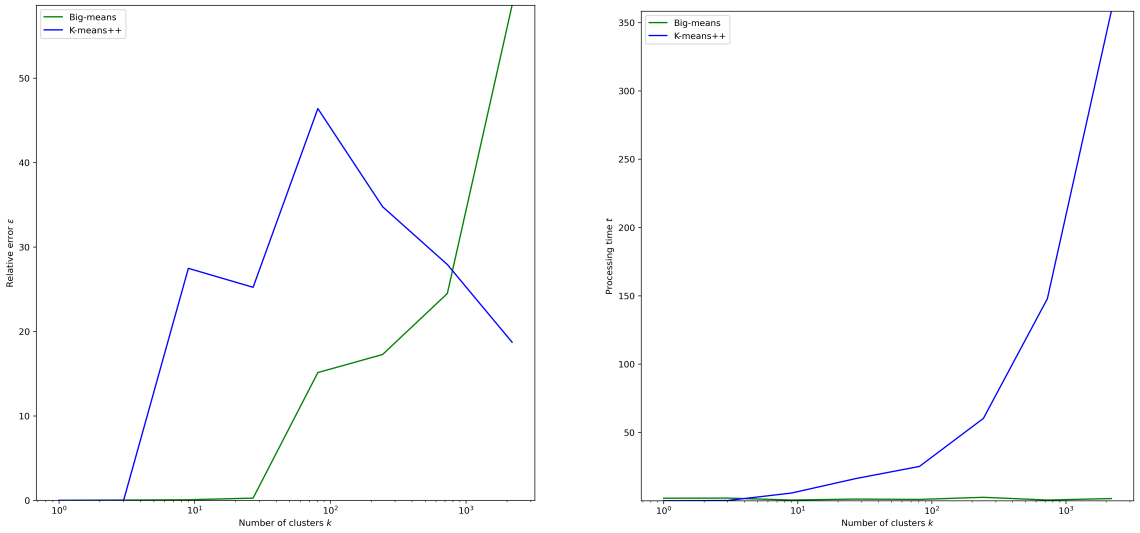
(d) (e) (f)

Figure 3.5: In 3.5a, 3.5d: the original datasets X_4 (999.69) and X_5 (6223.48); in 3.5b, 3.5e: the results of running K-means++ on X_4 (999.68) and X_5 (9825.219); in 3.5c, 3.5f: the results of running Big-means on X_4 (1017.59) and X_5 (6324.45)



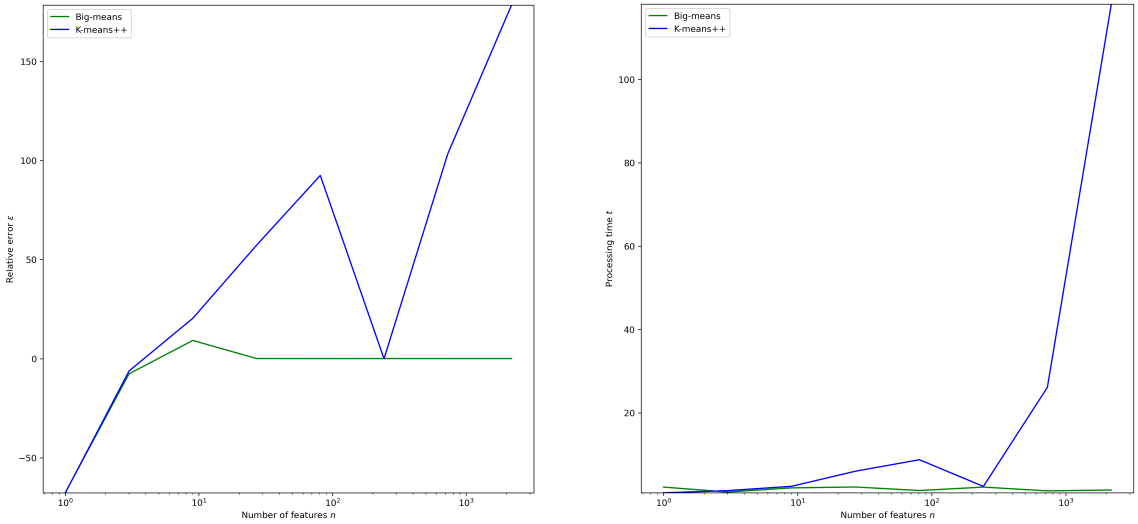
(a) (b)

Figure 3.6: Comparison of the performances of Big-means and K-means++ with respect to the variable number of points in a synthetic dataset



(a) (b)

Figure 3.7: Comparison of the performances of Big-means and K-means++ with respect to the variable number of clusters in a synthetic dataset



(a) (b)

Figure 3.8: Comparison of the performances of Big-means and K-means++ with respect to the variable number of features in a synthetic dataset

Chapter 4

HIGH-PERFORMANCE HYBRID ALGORITHM FOR GLOBAL MINIMUM SUM-OF-SQUARES BIG DATA CLUSTERING

4.1 Introduction

4.1.1 Motivation

The results of Chapter 3 established that Big-means reached state-of-the-art performance among the competitive MSSC algorithms on the problem of big data clustering [Mus+23]. This success is mainly due to the decomposition principle, which is at the heart of Big-means [Mus+23]. However, decomposition not only serves as the algorithm's cornerstone but also facilitates its efficient parallelization. Therefore, an extensive optimization of Big-means through parallelization should lead to promising further improvements. In this chapter, following the goal of maximizing the performance of Big-means for big data, we continue to optimize Big-means along this critical dimension.

This chapter introduces HPClust, a set of novel parallel approaches to the MSSC problem, tailored for infinitely tall big data. By utilizing modern high-performance computing techniques, HPClust enhances key clustering metrics: effectiveness, computational efficiency, and scalability. In contrast to vanilla data parallelism, which only accelerates processing time through the MapReduce framework, our approach unlocks superior performance by leveraging the multi-strategy competitive-cooperative parallelism and intricate properties of the objective function landscape. Unlike other available algorithms that struggle to scale, our algorithm is inherently parallel in nature, improving solution quality through increased scalability and parallelism, and outperforming even advanced algorithms designed for small and medium-sized datasets. Our evaluation of HPClust, featuring four parallel strategies,

demonstrates its superiority over traditional and cutting-edge methods by offering better performance in the key metrics. These results also show that parallel processing not only enhances the clustering efficiency, but the accuracy as well. Additionally, we explore the balance between computational efficiency and clustering quality, providing insights into optimal parallel strategies based on dataset specifics and resource availability. This research advances our understanding of parallelism in clustering algorithms, demonstrating that a judicious hybridization of advanced parallel approaches yields optimal results for MSSC. Experiments on synthetic data further confirm HPClust's exceptional scalability and robustness to noise.

Parallel processing in big data clustering algorithms presents a critical and frequently overlooked aspect. Most approaches that have been discovered in the literature are limited to only data parallelism, which is usually implemented using the MapReduce model. Meanwhile, more sophisticated parallel strategies are either not investigated or not applicable to the big data clustering algorithms available in the literature.

Four parallel approaches — inner, competitive, cooperative, and hybrid — are proposed to tackle the MSSC-ITD problem. The inner parallel method involves parallelizing distance evaluations in the K-means local search applied within each sequential clustering subproblem, offering scalability in the subproblem size. The competitive strategy implements concurrency at the subproblem level, maximizing diversity in initial clustering solutions. The cooperative approach simultaneously processes clustering subproblems, maximizing exploration by continuously selecting the best solution and capitalizing on it. The hybrid strategy combines the last two into a multi-strategy competitive-cooperative approach, aiming for an optimal exploration-exploitation trade-off in MSSC-ITD solutions.

The name HPClust can be interpreted in two ways, both reflecting the algorithm's key strengths. Firstly, "High-Performance Clustering" highlights the algorithm's computational efficiency, speed, and ability to scale through parallelism, making it a high-performance solution for clustering tasks. Alternatively, "Hybrid Parallel Clustering" emphasizes the innovative combined parallel clustering strategy employed by HPClust, which leverages the strengths of different parallel approaches to achieve superior performance. This hybrid strat-

egy sets HPClust apart as a winning solution in the field of parallel clustering algorithms.

Notably, our algorithm boasts a significant conceptual advantage as one of the few clustering algorithms that is inherently parallel in nature. This allows it to improve solution quality through increased scalability and parallelism, setting it apart from other algorithms that may struggle to scale. Moreover, our algorithm is capable of competing with advanced clustering algorithms designed for small and medium-sized datasets, demonstrating its versatility and robustness. Unlike other algorithms where parallelism is a forced add-on, our algorithm's parallel nature is an intrinsic property that enables seamless scalability.

While other approaches to clustering often rely solely on data parallelism, our approach utilizes a combination of more advanced and sophisticated parallelism types. Data parallelism involves dividing the dataset into smaller chunks and processing each chunk simultaneously on different processors, but only brings advantages in processing time. In contrast, task parallelism (functional parallelism) enables us to execute different tasks or functions of the clustering algorithm in parallel, allowing for more flexibility and effectiveness when merging their results. Furthermore, hybrid parallelism combines these approaches, allowing us to leverage the strengths of each to achieve better results. Unlike other parallel approaches that only focus on scaling clustering in the data space without guarantees on solution quality, our approach leverages the strengths of different parallelism types by combining data parallelism with task parallelism and hybrid parallelism, achieving better results. This integrated approach sets our method apart from others, which often rely on a single type of parallelism, and enables us to deliver higher-quality clustering solutions and scalability in big data clustering.

As previously noted, MSSC algorithms are known to be computationally intensive due to their NP-hard complexity [Alo+09]. The NP-hardness of MSSC is heavily exacerbated by big data contexts. High-Performance Computing (HPC) technologies, including supercomputers and computer clusters, offer a robust platform for tackling such complex problems. By distributing the data across multiple nodes, computers, or processors, parallel processing enables scalable and efficient handling of big data. This approach leverages the combined

computational power of multiple computing resources, allowing for faster and more effective execution of MSSC algorithms on massive datasets.

Also, we provide a comprehensive review of various parallel and high-performance computing techniques used for big data clustering and indicate their strengths and weaknesses. We pinpoint the intricacies involved in the process of applying these approaches to HPClust, as well as exhibit the obtained insights in form of a tutorial on applying parallel and high-performance computing technologies to the problem of big data clustering.

4.1.2 Chapter structure

The chapter has the following structure. Section 4.2 surveys the key developments and strategies in the field of parallel clustering algorithms. Section 4.3 presents the proposed HPClust algorithm, while Section 4.4 describes its various parallel strategies. Section 4.5 provides an overview of modern high-performance techniques for optimizing big data clustering algorithms, highlighting key nuances and considerations in the implementation details of the HPClust algorithm. Section 4.6 describes our experimental setup and its methodology. Section 4.7 provides a detailed analysis and interpretation of our experimental findings, along with insights into trade-offs. Section 4.8 offers practical guidelines for selecting the optimal parallel strategy for HPClust, aimed at big data clustering practitioners. Finally, Section 4.9 concludes our work and identifies promising future research directions.

4.2 Related works

In the field of big data clustering, many methods have been created that work in parallel and distribute the workload to handle the difficulties presented by the large size, complex dimensions, and real-time nature of big data. Parallelism and distributed computing appear as two prominent techniques for big data clustering.

Usually, parallel processing in clustering algorithms involves dividing the data into smaller subsets, clustering them simultaneously on multiple processors, and aggregating these partial results into a global solution. This helps in reducing the computation time and makes the

clustering process much more efficient. It is usually used when the data is too large to fit into memory or the computation time is a bottleneck.

Distributed computing, on the other hand, involves the distribution of big data across multiple machines. Clustering is then performed in a distributed manner using frameworks like Apache Hadoop or Apache Spark [Moh+16]. By distributing the data and computations, processing time is reduced, and scalability is achieved. This approach is useful when a dataset is unacceptably large to be stored and processed on a single machine.

K-means [HJ82] algorithm with the Forgy initialization [Pen99] is a commonly used traditional clustering method due to its simplicity and effectiveness. However, its application to big data can pose problems due to its high time complexity, which is $\mathcal{O}(m \cdot n \cdot k)$ for a single iteration, and the need to store all data in memory. The pseudocode of the Forgy K-means clustering method is provided in Algorithm 6.

Algorithm 6: Forgy K-means Clustering

Function `Forgy_k_means`(X, k)**:**
  Initialization:
  Randomly select k instances from X to serve as the initial centroids $C = (c_1, \ldots, c_k)$;
  Iterative Optimization:
  repeat
    Assign each $x \in X$ to the nearest centroid in C;
    Update each $c_i \in C$ to the mean of points assigned to c_i;
  until *centroids C are unchanged or maximum iterations reached*;
  Cluster Assignment:
  Assign each $x \in X$ to its closest centroid in C, forming Y;
  return $C, \quad Y$

To circumvent the time complexity limitations of traditional clustering approaches, like Forgy K-means, some parallel and distributed clustering algorithms have been suggested in the literature. The MapReduce framework is by far the most popular approach to scale clustering in the data space [DG08]. Zhao et al. [ZMH09] implemented a distributed version of K-means according to the MapReduce concept that led to a significant speed-up compared

to the sequential version without any guarantees on the clustering solution quality.

A widely adopted method to handle large datasets that cannot be accommodated entirely in RAM is the Minibatch K-means algorithm [Scu10]. It is an online version of the K-means algorithm that employs random subsets, or minibatches, of a dataset during each iteration to update the current solution. While this technique significantly accelerates computation time, it sacrifices the clustering quality since it exerts no control over the solution updates across iterations. Also, Minibatch K-means is an inherently sequential algorithm, amenable to only data parallelism.

Bahmani et al. [B+12] developed a scalable version of K-means++ that merges the advantages of K-means++ and Minibatch K-means. However, our experimental evaluation on large real-world datasets in Chapter 3 showed that K-means||, while being on par with K-means++ in speed, is significantly worse that K-means++ with respect to solution quality.

Alguliyev et al. proposed an innovative approach in their study, where they introduced the Parallel Batch K-means For Big Data Clustering (PBK-BDC) algorithm [AAS21]. This algorithm partitions large datasets into smaller segments, clusters them with the help of K-means, and aggregates the resulting cluster centers into a final pool. The algorithm then clusters the pool using K-means again. The pseudocode for the PBK-BDC algorithm can be found in Algorithm 7. Notably, PBK-BDC is one of the most prominent partition-based clustering algorithms. In the original paper, the authors empirically evaluated PBK-BDC and found that it outperformed the classical K-means algorithm [AAS21]. However, this evaluation did not compare PBK-BDC to other advanced algorithms for clustering large datasets, leaving room for further research.

Mohebi et al. [Moh+16] conducted a comprehensive review of various parallel algorithms and concluded that the field of big data clustering using parallel computing is still in its emergent stage and offers significant scope for further research. They observed that parallel data processing can potentially reduce the clustering time of large datasets, but it may also have an adverse impact on the quality and performance of clustering. Thus, the primary objective of research in this area should be to achieve an optimal balance between quality

Algorithm 7: PBK-BDC Clustering Method

Function PBK_BDC(X, k, s)**:**
  Initialization:
  Divide the dataset X into segments, each containing p elements;
  foreach *segment* C_i **do**
    Apply K-means clustering to C_i to derive new centroids $C_{i,\text{new}}$;
    Incorporate $C_{i,\text{new}}$ into the cooperative centroid repository P;
  end
  Execute K-means clustering on repository P to secure the ultimate centroids C_{final};
  Map every data point in X to its closest centroid in C_{final}, establishing the ultimate cluster mappings Y;
  return $C_{\text{final}}, \quad Y$

and speed of clustering for big data applications.

Our proposed HPClust algorithm, utilizing advanced parallel processing techniques and intelligent sample selection, seeks to fill the gaps in the field. HPClust proves that advanced parallel strategies and careful algorithm design may optimize both the efficiency and effectiveness of clustering algorithms simultaneously, while maintaining exceptional scalability across various data scales.

4.3 Proposed algorithm

We propose HPClust, an array of parallel heuristic approaches for solving the MSSC-ITD problem via high-performance computing techniques. The algorithm's main idea is to apply the problem decomposition technique, letting each parallel worker iteratively process a sequence of subproblems, and intelligently combine the obtained partial results into a single global clustering solution.

Each parallel worker w operates by sequentially clustering incoming samples of a large dataset. It begins by randomly selecting a small sample S of size s from X, and uses the K-means++ algorithm to obtain the initial configuration of centroids C. The worker then clusters each new incoming sample by the K-means algorithm using the best set of centroids

C_w (or C_{best}) obtained from all previously processed samples for the current worker (or among all parallel workers), called the incumbent solution. The incumbent solution is chosen based on the objective function value (1.1) obtained on a sample. This "keep the best" principle allows the algorithm not to lose information about the best local minimum obtained so far, and more iterations can only lead to further improvements.

HPClust solves the issue of degenerate clusters (also known as empty clusters) by reinitializing them with K-means++ when all data points are reassigned to other clusters. This introduces new cluster centers, enhancing the overall clustering solution and increasing opportunities to minimize the objective function. Also, introducing new samples in each iteration perturbs the incumbent solution, injecting variability into the clustering outcomes.

When a stop condition is reached by any parallel worker (e.g., a time limit or maximum number of processed samples), the algorithm selects the final centroid set C by choosing the solution obtained by the worker with the lowest incumbent sample objective function. Then, HPClust assigns data points of the entire dataset to their closest cluster centers in the final centroid set C. However, this final assignment step may be omitted if only the final centroids or a limited set of assignments are required.

HPClust's iteration time complexity is $\mathcal{O}(s \cdot n \cdot k)$ (where k is the number of clusters). The algorithm's scalability can be fine-tuned by selecting suitable sample sizes and counts. By processing smaller subsets of the data in each iteration, the computational demands are substantially reduced. Additionally, employing random subsets of the data during each iteration and periodically re-initializing the centroids of degenerate clusters prevents the algorithm from being trapped in suboptimal solutions. This allows the algorithm to explore different parts of the objective function's landscape, potentially finding better solutions than a single application of the K-means algorithm.

4.4 Parallel strategies

The HPClust algorithm is designed to be highly parallel in nature. Four different parallel strategies can be employed:

1. Inner parallelism (HPClust-inner): Employs parallel clustering at the implementation level of K-means and K-means++, processing individual data samples sequentially while parallelizing the calculation of minimum distances;

2. Competitive parallelism (HPClust-competitive): Independent workers process individual data samples in parallel, each using its own previous best centroids C_w for initialization, and the best solution is selected when the stopping criterion is met. A pseudocode of the HPClust-competitive algorithm is shown in Algorithm 8;

3. Cooperative parallelism (HPClust-cooperative): Workers share information on best solutions and use the best set of centroids C_{best} obtained from all previous iterations across every worker, initializing each subsequent sample using the cooperative knowledge. A pseudocode of the HPClust-cooperative algorithm is provided in Algorithm 9;

4. Hybrid or competitive-cooperative parallelism (HPClust-hybrid): Combines competitive and cooperative strategies, initially utilizing diversity through competitive parallelism for a duration of T_1 seconds or N_1 iterations, and then capitalizing on the most successful evolved form through cooperative parallelism for an additional T_2 seconds or N_2 iterations. A pseudocode of the HPClust-cooperative algorithm is presented in Algorithm 10.

The goal of the hybrid mode is to leverage the advantages of both competitive and cooperative approaches, ensuring diversity and exploiting the most successful solutions. Flowcharts for the competitive and cooperative strategies are provided in Figures 4.1 and 4.2.

The HPClust algorithm source code, including implementations of various parallel strategies, is available at https://github.com/rmusab/hpclust.

Our study focuses on the efficiency of parallel interaction strategies, assuming equal access to the full-sized dataset and independent sampling, without exploring distributed data storage optimizations, which are left for a separate study.

Algorithm 8: Competitive HPClust Clustering

Function `HPClust_competitive(`X, k, s, T`):`
  Input : Feature vectors $X = \{x_1, \ldots, x_m\}$;
    Desired number of clusters k;
    Sample size s;
    Maximum time T;
  Output: Centroids $C = \{c_1, \ldots, c_k\}$;
    Point-to-cluster assignments $Y = \{y_1, \ldots, y_m\}$.
  $C_w \leftarrow \{\varnothing_1, \ldots, \varnothing_k\}$ for each worker w;
  $\hat{f}_w \leftarrow \infty$ for each worker w;
  $t_w \leftarrow 0$ for each worker w;
  while $t_w < T$ *for any worker* w **do**
    for *each parallel worker* w **do**
      Draw a random sample S_w of size s from X;
      for *each centroid* c *in* C_w **do**
        if c *represents a degenerate cluster* **then**
          Reinitialize c using K-means++ on S_w;
        end
      end
      Compute new centroids $C_{\text{new},w}$ using K-means on S_w with initial centroids C_w;
      if $f(C_{new,w}, S_w) < \hat{f}_w$ **then**
        $C_w \leftarrow C_{\text{new},w}$;
        $\hat{f}_w \leftarrow f(C_{\text{new},w}, S_w)$;
      end
      $t_w \leftarrow t_w + 1$;
    end
  end
  $C_{\text{best}} \leftarrow$ Centroids of the worker with the smallest $\hat{f}_w$ value;
  $Y \leftarrow$ Assign each point in X to nearest centroid in C_{best};
  return $C_{\text{best}}, \quad Y$

Algorithm 9: Cooperative HPClust Clustering

Function `HPClust_cooperative(`X, k, s, T`):`

Input : Feature vectors $X = \{x_1, \ldots, x_m\}$;
Desired number of clusters k;
Sample size s;
Maximum time T;

Output: Centroids $C = \{c_1, \ldots, c_k\}$;
Point-to-cluster assignments $Y = \{y_1, \ldots, y_m\}$.

$C_w \leftarrow \{\varnothing_1, \ldots, \varnothing_k\}$ for each worker w;
$\hat{f}_w \leftarrow \infty$ for each worker w;
$t_w \leftarrow 0$ for each worker w;
while $t_w < T$ *for any worker* w **do**
  for *each parallel worker* w **do**
    Draw a random sample S_w of size s from X;
    $C_{\text{best}} \leftarrow$ Centroids of the worker with the smallest $\hat{f}_w$ value;
    for *each centroid* c *in* C_{best} **do**
      if c *represents a degenerate cluster* **then**
        Reinitialize c using K-means++ on S_w;
      end
    end
    Compute new centroids $C_{\text{new},w}$ using K-means on S_w with initial centroids C_{best};
    if $f(C_{new,w}, S_w) < \hat{f}_w$ **then**
      $C_w \leftarrow C_{\text{new},w}$;
      $\hat{f}_w \leftarrow f(C_{\text{new},w}, S_w)$;
    end
    $t_w \leftarrow t_w + 1$;
  end
end
$C_{\text{best}} \leftarrow$ Centroids of the worker with the smallest $\hat{f}_w$ value;
$Y \leftarrow$ Assign each point in X to nearest centroid in C_{best};
return $C_{\text{best}}, \quad Y$

Algorithm 10: Hybrid HPClust Clustering

Function `HPClust_hybrid`(X, k, s, $T_{Competitive}$, $T_{Cooperative}$)**:**
  Input : Feature vectors $X = \{x_1, \ldots, x_m\}$;
    Desired number of clusters k;
    Sample size s;
    Maximum time for competitive phase $T_{\text{Competitive}}$;
    Maximum time for cooperative phase $T_{\text{Cooperative}}$;
  Output: Centroids $C = \{c_1, \ldots, c_k\}$;
    Point-to-cluster assignments $Y = \{y_1, \ldots, y_m\}$.
  $C_w \leftarrow \{\varnothing_1, \ldots, \varnothing_k\}$ for each worker w;
  $\hat{f}_w \leftarrow \infty$ for each worker w;
  $t_w \leftarrow 0$ for each worker w;
  for *Phase in* (*Competitive*, *Cooperative*) **do**
    while $t_w < T_{Phase}$ *for any worker* w **do**
      for *each parallel worker* w **do**
        Draw a random sample S_w of size s from X;
        if *Phase* = *Cooperative* **then**
          $C_{\text{base}} \leftarrow$ Centroids of the worker with the smallest $\hat{f}_w$ value;
        else
          $C_{\text{base}} \leftarrow C_w$;
        end
        for *each centroid* c *in* C_{base} **do**
          if c *represents a degenerate cluster* **then**
            Reinitialize c using K-means++ on S_w;
          end
        end
        Compute new centroids $C_{\text{new},w}$ using K-means on S_w with initial centroids C_{base};
        if $f(C_{new,w}, S_w) < \hat{f}_w$ **then**
          $C_w \leftarrow C_{\text{new},w}$;
          $\hat{f}_w \leftarrow f(C_{\text{new},w}, S_w)$;
        end
        $t_w \leftarrow t_w + 1$;
      end
    end
  end
  $C_{\text{best}} \leftarrow$ Centroids of the worker with the smallest $\hat{f}_w$ value;
  $Y \leftarrow$ Assign each point in X to nearest centroid in C_{best};
  return $C_{\text{best}}, \quad Y$

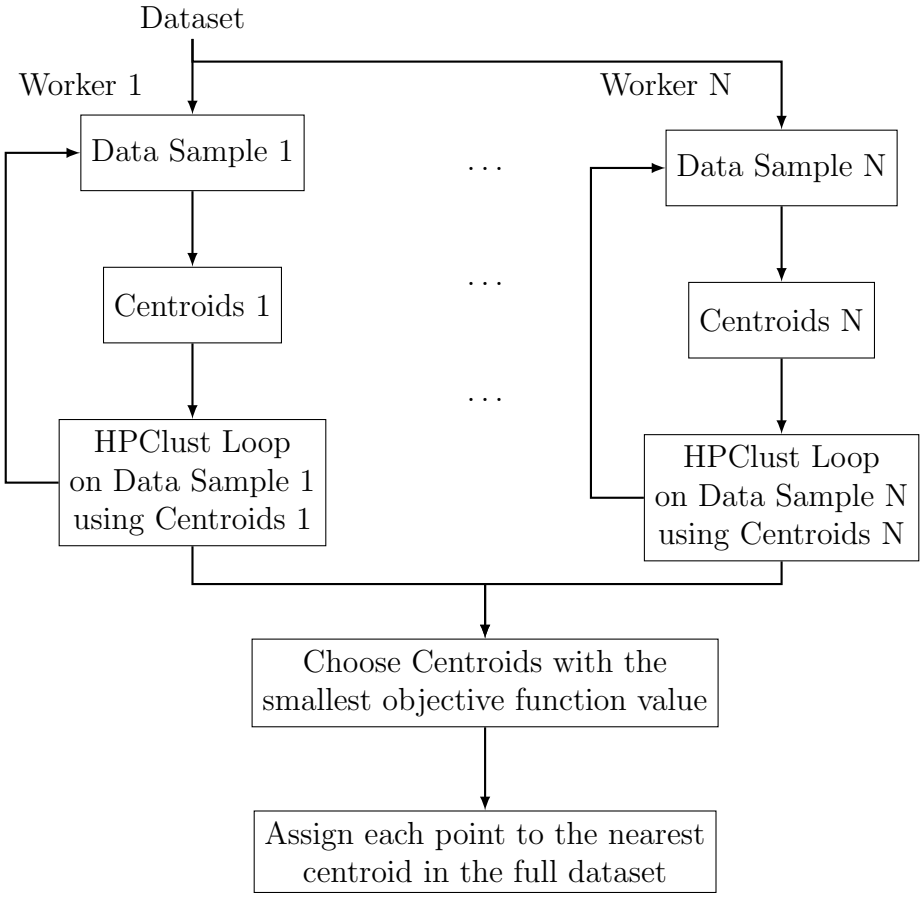


Figure 4.1: Flowchart of the HPClust algorithm with the competitive parallelism

4.5 High-performance techniques in HPClust

4.5.1 Analytical optimization

In the analytical optimization of computational algorithms, several high-performance computing techniques are relevant. These techniques represent algorithmic improvements or theoretical advancements applied at the abstract level of the algorithm itself.

- Parallel processing of iterations;

- Data sampling and partitioning;

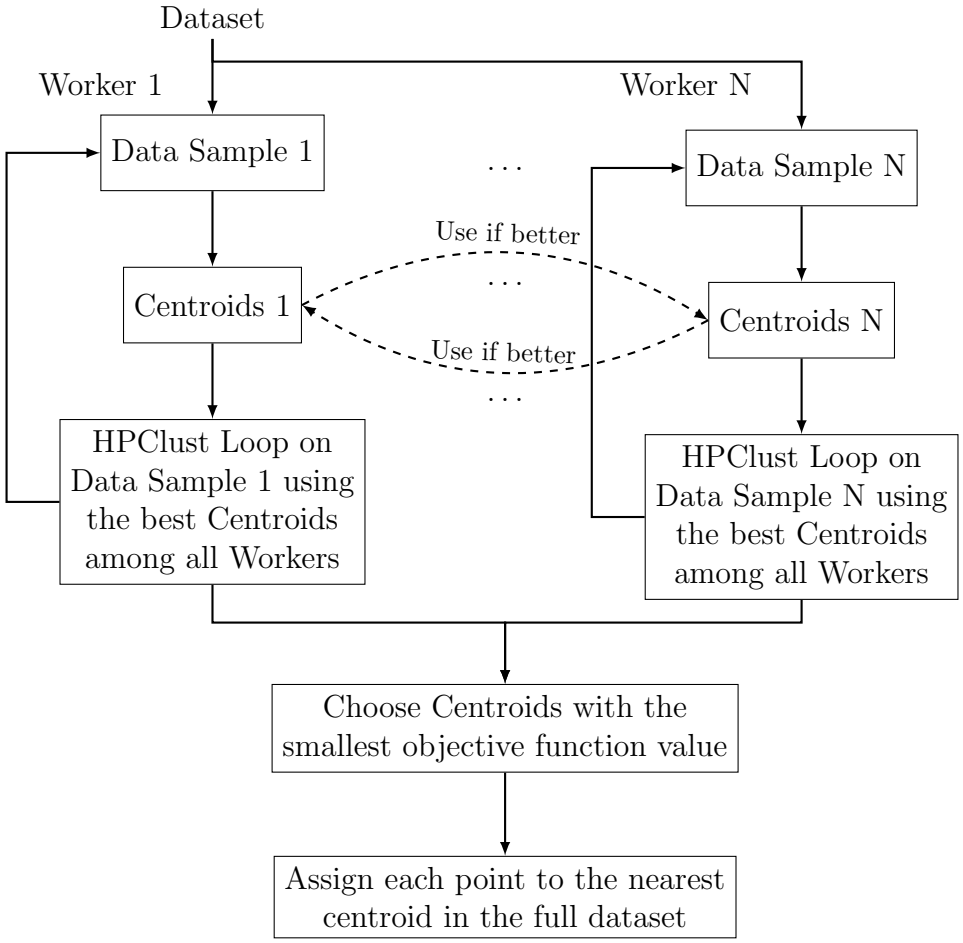


Figure 4.2: Flowchart of the HPClust algorithm using a cooperative parallel strategy

- Tuning the level of parallelism;
- Optimizing inter-process communication.

Parallel processing of iterations allows for simultaneous processing of multiple iterations. This strategy employs the execution of various instances of the algorithm on different subsets of data, significantly reducing the time required for convergence [CW00].

In relation to data management, HPClust can operate on subsets of data, allowing for a strategy of data partitioning. The initial dataset can be divided into smaller sections, each to be processed by an individual computing unit. This technique, known as data parallelism,

proves particularly useful when handling datasets that exceed the memory capacity of a single machine [DLS21].

The strategy of data sampling, wherein a random sample is selected from the dataset, can also be parallelized [He+15]. Especially in cases of extensive datasets, scanning the complete dataset becomes time-consuming. By distributing the dataset across multiple processors, each can sample a section of the data independently. Then, the resultant samples can be combined.

Tuning the level of parallelism to the specifics of a dataset can lead to significant performance improvements [SSE13].

Optimizing inter-process communication by designing an algorithm to minimize data transfer between processes can improve performance. Techniques such as compression, delta encoding, or other forms of data reduction can also be utilized [GSS96].

4.5.2 Nuances of parallelism in HPClust

The HPClust algorithm, a partitioning-based clustering method, is well-suited for parallelism across its key processes. Within its inner parallel variant, HPClust-inner, two primary operations — initialization and centroid updating — can be executed concurrently. Initially, the algorithm leverages K-means++ on a subset of data, calculating distances from points to centroids, which can be done in parallel due to the independent nature of these calculations.

During each K-means iteration, the algorithm updates centroids (denoted as C_{new}) by measuring distances from all points in the sample to these new centroids, thereby redefining clusters. This centroid update phase shares the parallelizable characteristic of the initialization phase.

Despite the benefits of parallel processing in speeding up these tasks, it introduces certain challenges, such as the need for effective load balancing across cores or processors to avoid inefficiencies like idle processors, especially when the sample size s is much smaller than the number of processors.

Moreover, implementing parallel computation in HPClust requires careful attention to

concurrency control to avoid race conditions — scenarios where the outcome depends on the order or timing of thread execution. In HPClust, threads may concurrently modify shared memory, such as updating centroid or data point memberships, potentially leading to inconsistent results.

To address these issues, synchronization mechanisms like locks, semaphores, or atomic operations are essential to ensure single-thread access to shared data, maintaining consistency and integrity. Optimizing the algorithm to reduce shared memory access can also help minimize race conditions. However, over-synchronization should be avoided as it can cause thread contention and decrease parallel efficiency.

For HPClust's parallel performance, it is important to achieve an optimal balance between data protection and computational speed. The aim is to improve computational speed through parallel processing without altering the clustering outcomes, maintaining consistency in results irrespective of the processor count. However, unlike other parallel clustering algorithms, this is not required for HPClust. Instead, HPClust can achieve higher accuracy by performing more iterations within a fixed time interval. This means that parallelism in HPClust improves not only efficiency but also accuracy.

Furthermore, the robustness of HPClust's parallel strategies is evident in centroid initialization, where allowing each worker to independently determine initial centroids helps overcome the challenges of poor initial selections, a known issue in K-means clustering. This feature emphasizes the importance of effective parallel design in maximizing HPClust's performance and accuracy.

4.5.3 Implementation-level optimization

To technically optimize the performance of HPClust on parallel or distributed computing systems, the following programmatic implementation-related techniques can be employed:

- Vectorized operations;
- SIMD instructions;

- Concurrent data structures;
- Distributed computing;
- Load balancing;
- Parallel random number generation;
- Parallel input/output (I/O).

Further, the utilization of vectorized operations also contributes to the optimization process. Libraries such as NumPy in Python and Armadillo in C++ offer the capacity for vectorized operations. The use of these operations across entire arrays, rather than individual elements, can lead to substantial speed increases. This is due to the reduction in loop overhead and more efficient utilization of CPU features [PBB22].

Simultaneously, modern CPUs provide support for SIMD (Single Instruction Multiple Data) instructions. With these, the same operation can be performed across multiple data points concurrently [Chh+08]. Vectorizing computations, such as distance calculations in the HPClust algorithm, allows for the exploitation of these instructions, resulting in significant speed gains.

Modern programming languages and libraries offer concurrent data structures, which are designed for safe use across multiple threads or processes. These structures can prevent race conditions and synchronization issues, contributing to the efficiency of parallel algorithms [SSP07].

For extremely large datasets that exceed the memory of a single machine, distributed computing frameworks such as Apache Hadoop or Apache Spark are beneficial. These frameworks facilitate the distribution of data and computation across several nodes in a cluster, accommodating larger datasets than would be possible on a single machine [DLS21].

Load balancing is a strategy to efficiently use computational resources, ensuring an even distribution of work across all threads or processes. This strategy may include the dynamic

assignment of tasks to processors based on their current workload. Alternatively, more sophisticated load balancing algorithms can be employed [JMN17].

The generation of random numbers, a function of the HPClust algorithm, can also be performed in parallel. Several techniques and libraries support parallel random number generation, maintaining independent and identically distributed numbers [BS14].

Finally, parallel I/O techniques can help alleviate the bottleneck caused by input/output operations such as reading data from disk or writing results back. A parallel file system or separate threads or processes performing I/O operations can facilitate this [May01].

To implement these parallel strategies, various libraries and frameworks can be utilized. OpenMP or MPI in C/C++, and multiprocessing in Python offer traditional approaches. For GPU-accelerated parallel computation, CUDA or OpenCL are typically used. However, for a balance between functionality and simplicity, one might also consider employing modern libraries such as Numba. Numba provides a just-in-time compiler for Python that is easy to use yet powerful. Mojo is another notable option, providing simple and efficient parallelization solutions with a focus on high-level, user-friendly interfaces. To take full advantage of modern hardware architectures, one could use optimized numerical libraries, such as Intel's Math Kernel Library (MKL) or cuBLAS for GPUs. These libraries provide highly optimized implementations of common mathematical operations, which can lead to significant speedups.

Numba [LPS15; Mar18] is a key instrument in high-performance computing, featuring optimization capabilities such as parallelization, multi-threading, and vectorization. These features are core strategies in performance optimization, transforming the execution speed of Python functions, loops, and numerical computations. Numba's dynamic generation of optimized machine code for both CPUs and GPUs further contributes to this performance boost, converging Python's usability and the speed of lower-level languages.

Numba's proficiency extends to CUDA support, facilitating the optimization of computational procedures through the use of NVIDIA GPUs. Moreover, it showcases seamless integration with Python's scientific stack, demonstrating compatibility with NumPy, SciPy,

and Pandas, thereby optimizing Python's computational efficiency. In the context of distributed computing, Numba's interplay with Dask, a parallel computing library in Python, introduces an additional level of optimization, enabling efficient large-scale computations. Therefore, Numba serves as a potent tool in scientific computing, optimizing the bridge between Python's user-friendly nature and the computational efficiency of lower-level languages.

4.5.4 Future optimization directions

Future optimization of the HPClust algorithm can leverage the following high-performance techniques:

- Dynamically adjusting the number of threads;
- Reducing communication overhead.

The number of threads can be adjusted dynamically, depending on the current system load and the size of the processed data subset, maximizing the use of CPU cores [SSE13].

The overhead in communication between different threads or processes is a major concern in parallel algorithms [GSS96]. Designing the algorithm to allow each thread or process to operate independently, reducing the need for communication, can address this.

4.6 Experimental setup

4.6.1 Hardware and software

The experiments were performed on a system running Ubuntu 22.04 64-bit, powered by an AMD EPYC 7663, with up to 16 cores used in our experiments. The system was equipped with 1.46 TB of RAM. The software stack consisted of Python 3.10.11 along with NumPy 1.24.3 and Numba 0.57.0. The Numba [Mar18] package was used to accelerate Python code execution and facilitate parallelism. The use of Numba is particularly advantageous for

these purposes due to its ability to compile Python code into machine code at runtime and its capabilities for executing code on multiple processors.

4.6.2 Competitive algorithms

We compare the performance of HPClust, equipped with different parallel strategies, to two benchmark algorithms: Forgy K-means [Pen99] and PBK-BDC [AAS21]. Forgy K-means serves as a basic lower benchmark, representing a simple and straightforward approach. On the other hand, PBK-BDC is an advanced upper benchmark, which represents the most optimized big data clustering algorithm available in the literature [AAS21].

4.6.3 Datasets

The experiments are conducted on 23 datasets: 19 are publicly available (detailed in Table 3.1 and Table 3.2), and 4 are normalized. These datasets, which are numerically based and have no missing values, vary significantly in size, from 7,797 to 10,500,000 instances, and feature 2 to 5,000 attributes. This variety ensures testing of HPClust's adaptability across different data scales. Additionally, we align our methodology with Karmitsa et al. [KBT18] for comparative analysis.

4.6.4 Experimental design and evaluation metrics

Each dataset undergoes clustering n_{exec} times into k clusters of varying number: 2, 3, 5, 10, 15, 20, 25. Each execution of an algorithm on a pair (X, k) is considered an experiment. We assess each experiment by measuring the resulting relative error (ε), CPU time (t), and baseline convergence time ($\bar{t}$). The relative error reveals the performance relative to historical bests: $\varepsilon = 100 \cdot (f - f^*)/f^*$. Sometimes, a relative error may yield a negative value, which actually indicates an even more impressive performance by the algorithm, surpassing expectations.

For the parallel multi-worker HPClust versions, t is defined to be the time of the last

change of the incumbent solution C_w for the worker w that achieved the best value of the objective function $f(C_w, S_w)$ on some sample S_w. For a fixed pair (X, k), baseline time $\bar{t}$ of algorithm A is defined to be the time of achieving the baseline sample objective value $\overline{f}_s$:

$$\overline{f}_s = \max_{j \in \{1,2,3,4\}} med\left(f_s^*(A_j)\right) \tag{4.1}$$

where the set of algorithms $\{A_j\}_{j=1}^4$ represents the HPClust versions, and $f_s^*(A_j)$ is the best value of the objective function evaluated on a sample using algorithm A_j. The median $med(\cdot)$ is calculated across n_{exec} executions of algorithm A_j for the fixed pair (X, k). This allows to cancel adverse outliers in the computations. Then, to capture the performance of the worst algorithm, the maximum of the medians is calculated. In the parallel multi-worker versions of HPClust, $\bar{t}$ represents the time t in seconds taken by the worker w that achieves the baseline sample objective value f_s^* before any other worker.

It is more objective to use the time metric $\bar{t}$ rather than the straightforward total clustering time t. This is because $\bar{t}$ helps to avoid certain undesirable situations not captured by the simple time metric t. For instance, a clustering algorithm might significantly improve the solution in a brief initial period of the total clustering time, while the remainder is spent achieving only marginal and insignificant improvements. Our time metric, $\bar{t}$, is designed to primarily reflect the duration of this initial, rapid improvement phase.

4.6.5 Hyperparameter selection

We set a maximum CPU time limit T and stop the K-means clustering process if iterations exceed 300 or the improvement between two consecutive objectives is less than 10^{-4}. For K-means++, we consider three candidate points for sampling each new centroid.

Sample sizes are optimized based on preliminary tests to ensure no further adjustments improve performance. The specific values of T and n_{exec} can be found in the detailed tables of experimental results included in the supplementary material. For each pair (X, k), the choice of parameters T and n_{exec} mostly match the values used in the experiments of Chapter 3.

These values can also be found in Appendix B.

4.6.6 Preliminary experiments

Preliminary experiments helped establish baselines and optimize parameters for the subsequent experiments. In the first preliminary experiment, the number of employed CPUs (workers) was varied in the range {2, 4, 8, 12, 16}, and the resulting values of the relative accuracy ε and CPU time t were measured 3 times for every pair (X, k). We observed that the results were quite robust with respect to the CPU number, so we believed that this choice of the execution number was enough for obtaining statistically significant results for this set of preliminary experiments. The execution numbers for the main experiment are much larger and match the values used in Chapter 3. The outcomes of the first set of preliminary experiments allowed us to determine 8 as the optimal number of parallel workers to use by HPClust in the subsequent experiments. In this context, the optimal selection means that this choice achieves the best balance between the solution quality and execution time simultaneously for all the considered algorithms, allowing for further fair comparison under equal conditions. Figures 4.3a and 4.3b depict the results.

In the next set of preliminary experiments, four parallel HPClust versions were run over all the datasets according to the methodology described above. Then, the baseline sample objective values $\overline{f}_s$ were computed according to definition (4.1). These values served as baselines in the main experiments.

The remaining preliminary experiments established the optimal choice of parameters for the hybrid parallel strategy of HPClust. For each dataset and number of clusters (X, k), the bounding time T was split into 30 intervals of length $\delta = T/30$. At each intermediate node $t_1 \in \delta, 2\delta, \ldots, i\delta, \ldots, 29\delta$ (excluding the endpoints) in the split, the hybrid parallel approach was executed three times. The bounding values T_1 and T_2 of the competitive and cooperative phases were set as $T_1 = t_1$ and $T_2 = T - t_1$, respectively. Finally, the split yielding the optimal median accuracy value is recorded.

4.6.7 Main experiments

The main experimental results are displayed using a special table format. Each algorithm and pair (X, k) originate a series of n_{exec} experiments. Each series has a minimum, median, and maximum resulting values of relative accuracy and time, which are calculated across n_{exec} executions of the algorithm on configuration (X, k). The means of these metrics across the values of k for each dataset are displayed in the corresponding columns of the presented tables. Tables 4.1 and 4.2 provide a comparison of the proposed HPClust parallel strategies, while Tables 4.3 and 4.4 compare the best HPClust parallel strategy with the selected competitive algorithms.

For instance, for a particular algorithm, we have the following entry in a table: ISOLET #Succ = 6/7; Min = 0.01; Median = 0.24; Max = 0.59. In this case, the ratio 6/7 indicates that for each of the 7 different values of $k \in \{2, 3, 5, 10, 15, 20, 25\}$, we performed a series of runs for each of the compared algorithms. For each fixed choice of (X, k), the corresponding series consists of $n_{exec} = 15$ independent runs of each algorithm. Thus, for each dataset, we have 7 series of runs for each of the compared algorithms, with each series containing 15 independent results. The number 6 in the #Succ ratio 6/7 indicates that the median objective function values for 6 out of 7 series of runs of this algorithm were lower than the median objective function values in the corresponding series of all other algorithms.

The means in the final rows of these tables highlight overall performance across datasets. The best results for each metric and dataset pair are bolded, indicating top algorithm performance. Success is indicated when an algorithm's median performance on a series of executions for a value of k outperforms or matches the best result among all the algorithms for this series.

4.6.8 Scaling experiment

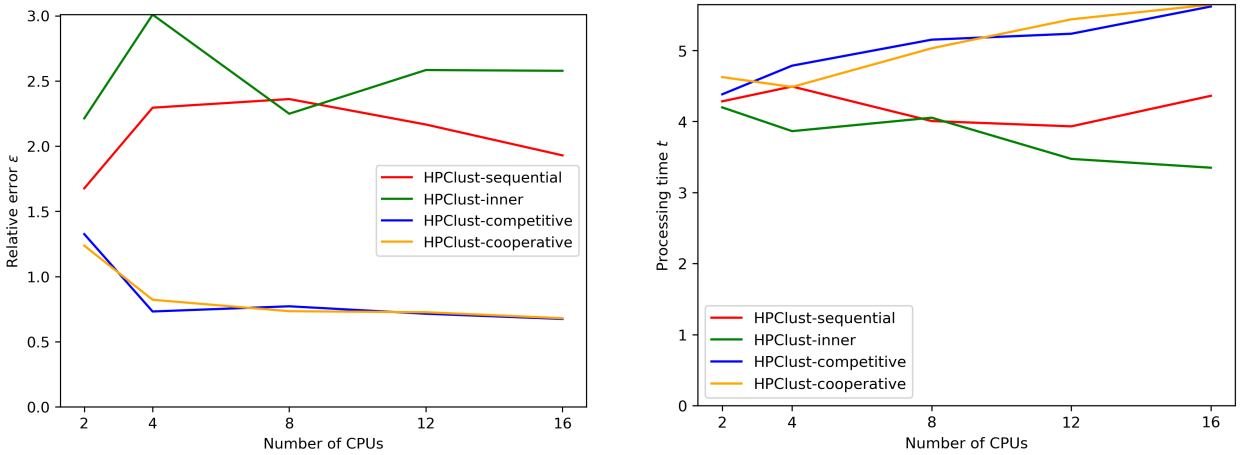
Additionally, we conducted an experiment to demonstrate the scalability of our proposed HPClust strategies. We generated a synthetic dataset with 10 features comprising 10 Gaus-

sian blobs uniformly distributed within the box $(-40, 40)$, each with a randomly sampled standard deviation between $(0, 10)$. The number of points was varied according to the law $m = 3^{i+7}$, where $i = 0, ..., 8$. For each i, we performed 10 execution repetitions for each algorithm and recorded the results. We employed a sample size of $s = \min\{5000, m - 1000\}$ and a processing time limit of $T = 3.0$ seconds for the HPClust and PBK-BDC algorithms. For HPClust-hybrid, we used a naive time split of $T_1 = T_2 = T/2$ to avoid additional optimization. To introduce noise, we added 500 random points uniformly distributed within the box $(-50.0, 50.0)$ to each synthetic dataset. This experiment allowed us to assess the scalability of our algorithms under varying dataset sizes.

4.7 Experimental results and discussion

4.7.1 Performance evaluation

The results of the first set of preliminary experiments, illustrated in Figures 4.3a and 4.3b, determined the optimal number of CPUs for subsequent experiments, setting the stage for further investigation. As anticipated, the fully sequential strategy (HPClust-sequential) displayed no significant correlation with the number of parallel processors employed. The HPClust version with inner parallelism demonstrated a reduction in processing time with an increase in the number of CPUs, while the accuracy remained independent of the CPU count. In contrast, both the HPClust-competitive and HPClust-cooperative strategies exhibited an improvement in clustering accuracy as the number of CPUs increased. However, this accuracy gain came at the expense of increased processing time for these versions of HPClust. We attribute this observation to the need for coordination among multiple processors and the technical complexities introduced by Numba, such as parallel access to shared memory locations by multiple workers. Upon closer examination of the scores, we determined that utilizing 8 CPUs strikes the optimal balance between processing time and resulting accuracy across all algorithms on our machine. Thus, this choice of the CPU count was used in all the subsequent experiments.



(a) Median accuracy values

(b) Median processing times

Figure 4.3: Comparative results of the algorithms with respect to the number of employed CPUs averaged across all the datasets

Other preliminary experiments were straightforward. They allowed to obtain the necessary optimal values of the parameters for the main set of experiments.

A summary of the results of the main experiments are provided in Tables 4.1, 4.2, 4.3, and 4.4. The total number of conducted main experiments reached $22,098$. Full details of the main experiments' results are provided in Appendix B.

As Table 4.1 demonstrates, the HPClust-competitive, HPClust-cooperative, and HPClust-hybrid strategies markedly boost overall clustering quality, achieving results that are up to three times better than HPClust-inner.

The HPClust-competitive approach showed a slight edge in average clustering quality compared to HPClust-cooperative, likely due to comprehensive initializations that mitigate K-means' sensitivity to initial conditions. The analysis highlights a trade-off between extensive local optimization with a single start point and multiple initializations. The experiments suggest that multiple initializations, persistently processed by the competitive method, lead to better outcomes than the cooperative method's focus on a single initialization. This

Table 4.1: Resulting relative clustering accuracies ϵ (%) for the proposed parallel HPClust strategies

Dataset	HPClust-inner				HPClust-competitive			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	0/7	0.07	0.21	0.34	3/7	**0.0**	**0.07**	0.18
HEPMASS	0/7	0.08	0.22	0.66	3/7	0.03	**0.07**	**0.19**
US Census Data 1990	2/7	0.92	3.13	35.87	3/7	0.48	**1.48**	**2.89**
Gisette	0/7	-0.43	-0.37	-0.19	2/7	-0.44	-0.38	-0.32
Music Analysis	3/7	0.41	0.91	6.24	4/7	0.43	**0.74**	**1.67**
Protein Homology	3/7	**0.15**	0.91	2.32	1/7	0.41	**0.88**	2.03
MiniBooNE Particle Identification	2/7	-0.03	0.51	402305.65	1/7	-0.07	**-0.0**	719099.04
MiniBooNE Particle Identification (normalized)	1/7	0.2	0.54	101.63	2/7	0.2	0.55	**1.1**
MFCCs for Speech Emotion Recognition	2/7	0.14	0.64	1.95	1/7	0.11	0.34	**0.76**
ISOLET	0/7	0.15	0.68	1.72	1/7	0.04	**0.23**	0.63
Sensorless Drive Diagnosis	1/7	-0.32	1.25	31.06	2/7	**-0.43**	**-0.27**	12.2
Sensorless Drive Diagnosis (normalized)	1/7	0.4	3.03	9.69	4/7	0.31	**1.06**	**3.26**
Online News Popularity	2/7	0.7	2.36	14.39	2/7	0.69	1.65	**3.74**
Gas Sensor Array Drift	2/7	0.15	3.24	12.29	2/7	**-0.05**	1.78	**3.77**
3D Road Network	2/7	0.04	0.4	1.24	2/7	**0.03**	0.22	1.06
Skin Segmentation	1/7	0.04	2.91	9.72	2/7	-0.05	1.05	4.36
KEGG Metabolic Relation Network (Directed)	3/7	-0.08	1.55	34.13	2/7	**-0.42**	**0.24**	**2.5**
Shuttle Control	1/8	0.17	5.68	41.76	2/8	-0.01	2.32	12.58
Shuttle Control (normalized)	1/8	0.89	2.81	17.98	0/8	**0.69**	1.79	**4.07**
EEG Eye State	3/8	0.54	0.79	7.15	2/8	0.53	0.56	119444.14
EEG Eye State (normalized)	0/8	**-0.06**	2.4	31.49	6/8	**-0.06**	**0.01**	67.39
Pla85900	0/7	**0.07**	0.37	1.7	3/7	**0.07**	**0.2**	0.73
D15112	2/7	0.1	0.48	1.76	3/7	0.08	**0.14**	**0.4**
Overall Results	32/165	0.19	1.51	17507.42	53/165	0.11	0.64	36463.84

Dataset	HPClust-cooperative				HPClust-hybrid			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	3/7	0.04	0.08	**0.16**	1/7	0.02	0.08	0.24
HEPMASS	0/7	0.04	0.16	0.57	4/7	**-0.01**	0.08	0.24
US Census Data 1990	1/7	0.45	1.64	4.37	1/7	**0.32**	1.72	3.17
Gisette	2/7	**-0.46**	-0.39	-0.32	3/7	-0.44	**-0.4**	**-0.34**
Music Analysis	0/7	0.4	0.83	2.68	0/7	**0.33**	0.85	2.24
Protein Homology	2/7	0.21	0.91	**1.81**	1/7	0.5	1.05	2.1
MiniBooNE Particle Identification	2/7	**-0.08**	**0.0**	0.37	2/7	-0.07	**-0.0**	**0.15**
MiniBooNE Particle Identification (normalized)	1/7	**0.19**	0.56	1.43	3/7	0.23	**0.51**	1.29
MFCCs for Speech Emotion Recognition	2/7	**0.1**	0.34	0.94	2/7	0.12	**0.33**	0.83
ISOLET	2/7	0.03	0.25	0.68	4/7	**0.01**	**0.23**	**0.59**
Sensorless Drive Diagnosis	2/7	-0.41	-0.21	11.82	2/7	-0.42	-0.21	**8.18**
Sensorless Drive Diagnosis (normalized)	1/7	**0.28**	1.39	4.0	1/7	0.38	1.34	3.81
Online News Popularity	2/7	0.56	**1.6**	7.79	1/7	**0.47**	1.69	7.86
Gas Sensor Array Drift	1/7	-0.04	0.91	4.05	2/7	0.06	**0.79**	3.99
3D Road Network	2/7	0.04	0.22	1.04	1/7	0.04	**0.21**	**0.88**
Skin Segmentation	2/7	**-0.22**	1.11	5.76	2/7	-0.02	**1.02**	**4.25**
KEGG Metabolic Relation Network (Directed)	1/7	-0.3	0.35	6.26	1/7	-0.29	0.25	23.7
Shuttle Control	4/8	**-0.14**	**1.55**	**4.76**	1/8	0.08	1.86	9.13
Shuttle Control (normalized)	2/8	0.81	2.22	4.97	5/8	0.71	**1.61**	4.49
EEG Eye State	1/8	0.53	0.57	**0.76**	2/8	**0.52**	**0.55**	6.05
EEG Eye State (normalized)	0/8	**-0.06**	**0.01**	56.9	2/8	**-0.06**	0.02	**16.2**
Pla85900	2/7	**0.07**	0.22	1.22	2/7	**0.07**	**0.2**	**0.58**
D15112	1/7	0.08	0.29	0.8	1/7	**0.07**	0.15	0.44
Overall Results	36/165	**0.09**	0.63	5.34	44/165	0.11	**0.61**	**4.35**

Table 4.2: Baseline convergence times $\bar{t}$ (in seconds) of the HPClust parallel strategies

Dataset	HPClust-inner				HPClust-competitive			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	2/7	**6.92**	16.1	24.82	1/7	12.78	17.45	24.12
HEPMASS	0/7	5.77	8.65	15.59	2/7	**2.35**	5.24	14.12
US Census Data 1990	0/7	0.24	0.63	2.07	1/7	0.19	0.53	1.63
Gisette	4/7	**3.27**	**4.4**	**6.38**	0/7	17.06	19.38	23.72
Music Analysis	0/7	**0.58**	**3.22**	7.19	0/7	1.44	4.02	7.89
Protein Homology	2/7	**0.79**	**1.71**	**3.18**	1/7	1.63	2.45	4.03
MiniBooNE Particle Identification	4/7	**0.46**	**1.05**	**2.38**	1/7	2.24	3.07	4.39
MiniBooNE Particle Identification (normalized)	3/7	**0.09**	**0.4**	**0.85**	1/7	0.28	0.49	0.91
MFCCs for Speech Emotion Recognition	1/7	**0.12**	**0.39**	**0.83**	1/7	0.29	0.57	0.96
ISOLET	0/7	**0.38**	**1.01**	2.93	0/7	0.85	1.88	3.84
Sensorless Drive Diagnosis	3/7	**0.14**	**0.29**	**0.9**	0/7	0.72	1.02	2.07
Sensorless Drive Diagnosis (normalized)	0/7	**0.02**	**0.09**	0.28	0/7	0.04	**0.09**	0.26
Online News Popularity	2/7	**0.09**	**0.27**	0.59	0/7	0.15	0.29	0.62
Gas Sensor Array Drift	0/7	**0.11**	**0.47**	1.68	0/7	0.29	0.69	1.64
3D Road Network	2/7	**0.08**	**0.23**	**0.49**	0/7	0.15	0.35	0.88
Skin Segmentation	0/7	0.03	0.07	0.18	1/7	**0.02**	0.05	**0.12**
KEGG Metabolic Relation Network (Directed)	0/7	**0.1**	**0.3**	**0.82**	1/7	0.26	0.46	0.97
Shuttle Control	0/8	0.1	0.32	0.87	0/8	0.09	0.29	0.74
Shuttle Control (normalized)	0/8	0.04	0.15	0.32	3/8	**0.02**	0.07	**0.2**
EEG Eye State	0/8	0.13	0.43	1.11	0/8	**0.06**	0.31	0.78
EEG Eye State (normalized)	0/8	**0.04**	**0.11**	0.74	0/8	0.06	0.12	0.34
Pla85900	0/7	0.07	0.64	1.42	1/7	**0.05**	0.35	1.14
D15112	0/7	0.06	0.42	1.11	3/7	0.06	**0.21**	**0.77**
Overall Results	23/165	**0.85**	**1.8**	**3.34**	17/165	1.79	2.58	4.18

Dataset	HPClust-cooperative				HPClust-hybrid			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	0/7	12.06	18.2	26.11	2/7	11.49	**16.03**	**23.92**
HEPMASS	4/7	2.92	**4.86**	**12.03**	0/7	2.6	6.69	16.98
US Census Data 1990	1/7	**0.14**	0.46	1.47	3/7	**0.14**	**0.45**	**1.41**
Gisette	0/7	16.95	18.99	23.07	0/7	16.99	19.33	23.71
Music Analysis	2/7	1.58	3.33	**6.99**	0/7	1.35	3.88	8.19
Protein Homology	2/7	1.73	2.52	4.23	0/7	1.83	2.91	4.43
MiniBooNE Particle Identification	0/7	2.19	2.83	4.4	1/7	2.05	3.05	4.33
MiniBooNE Particle Identification (normalized)	1/7	0.29	0.51	0.88	0/7	0.25	0.53	1.0
MFCCs for Speech Emotion Recognition	1/7	0.22	0.49	0.99	1/7	0.26	0.55	1.06
ISOLET	3/7	0.87	1.42	**2.88**	0/7	0.76	1.96	4.07
Sensorless Drive Diagnosis	0/7	0.62	1.05	2.0	1/7	0.72	1.05	1.95
Sensorless Drive Diagnosis (normalized)	3/7	0.04	**0.09**	**0.25**	2/7	0.04	**0.09**	0.27
Online News Popularity	2/7	0.14	0.28	**0.56**	1/7	0.14	0.29	0.71
Gas Sensor Array Drift	1/7	0.27	0.63	**1.62**	1/7	0.27	0.73	1.74
3D Road Network	0/7	0.18	0.33	0.87	1/7	0.16	0.37	1.18
Skin Segmentation	6/7	**0.02**	**0.04**	0.18	0/7	**0.02**	**0.04**	0.16
KEGG Metabolic Relation Network (Directed)	2/7	0.24	0.42	0.98	1/7	0.25	0.44	0.97
Shuttle Control	2/8	0.09	**0.21**	**0.59**	2/8	**0.08**	**0.21**	0.67
Shuttle Control (normalized)	5/8	**0.02**	**0.05**	0.21	0/8	**0.02**	0.07	0.26
EEG Eye State	2/8	0.07	0.23	0.89	3/8	0.08	**0.22**	**0.77**
EEG Eye State (normalized)	2/8	0.06	**0.11**	**0.33**	0/8	0.06	0.15	0.46
Pla85900	6/7	0.06	**0.24**	1.16	0/7	0.06	0.35	**1.11**
D15112	3/7	0.05	0.24	0.83	1/7	**0.04**	0.24	0.89
Overall Results	48/165	1.77	2.5	4.07	20/165	1.72	2.59	4.36

Table 4.3: Relative clustering accuracies ϵ (in %) resulting from the comparison of the hybrid HPClust strategy with the competitive algorithms

Dataset	HPClust-hybrid				Forgy K-means				PBK-BDC			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	3/7	0.02	**0.08**	**0.24**	4/7	**0.01**	0.17	1.37	0/7	0.67	1.74	3.28
HEPMASS	5/7	**-0.01**	**0.08**	**0.24**	2/7	0.02	0.18	0.63	0/7	0.63	1.45	3.21
US Census Data 1990	6/7	**0.32**	**1.72**	**3.17**	1/7	2.58	80.73	259.79	0/7	14.86	65.27	279.29
Gisette	0/7	-0.44	-0.4	-0.34	7/7	**-0.52**	**-0.48**	**-0.39**	0/7	-0.47	-0.42	-0.32
Music Analysis	1/7	0.33	0.85	**2.24**	6/7	**-0.01**	**0.47**	6.97	0/7	1.27	4.85	42.27
Protein Homology	4/7	**0.5**	**1.05**	**2.1**	3/7	14.84	14.91	15.09	0/7	4.98	20.63	48.21
MiniBooNE Particle Identification	4/7	**-0.07**	**-0.0**	**0.15**	3/7	2.62	19.52	110992.52	0/7	2.61	41006.39	110992.75
MiniBooNE Particle Identification (normalized)	2/7	0.23	**0.51**	**1.29**	5/7	**-0.02**	1.39	240.25	0/7	2.34	7.75	36.83
MFCCs for Speech Emotion Recognition	4/7	**0.12**	**0.33**	**0.83**	3/7	0.22	1.49	2.92	0/7	1.97	10.1	40.56
ISOLET	6/7	**0.01**	**0.23**	**0.59**	1/7	0.05	0.8	2.78	0/7	0.22	1.03	2.59
Sensorless Drive Diagnosis	7/7	**-0.42**	**-0.21**	**8.18**	0/7	122.75	162.37	183.78	0/7	149.77	162.36	215.62
Sensorless Drive Diagnosis (normalized)	6/7	**0.38**	**1.34**	**3.81**	1/7	1.3	6.21	26.96	0/7	4.49	11.24	48.1
Online News Popularity	5/7	**0.47**	**1.69**	**7.86**	2/7	7.76	14.93	33.83	0/7	15.31	37.76	93.96
Gas Sensor Array Drift	5/7	**0.06**	**0.79**	**3.99**	2/7	10.01	24.31	39.62	0/7	9.52	25.52	39.35
3D Road Network	1/7	0.04	**0.21**	0.88	6/7	**0.0**	0.23	**0.23**	0/7	2.67	40.65	159.28
Skin Segmentation	5/7	**-0.02**	**1.02**	**4.25**	2/7	2.17	9.02	21.32	0/7	7.46	20.55	71.1
KEGG Metabolic Relation Network (Directed)	6/7	**-0.29**	**0.25**	**23.7**	1/7	94.27	95.67	108.63	0/7	94.26	94.92	107.54
Shuttle Control	8/8	**0.08**	**1.86**	**9.13**	0/8	131.85	176.25	243.9	0/8	139.77	174.3	231.7
Shuttle Control (normalized)	6/8	**0.71**	**1.61**	**4.49**	2/8	2.63	16.59	74.13	0/8	8.54	31.94	105.37
EEG Eye State	7/8	**0.52**	**0.55**	**6.05**	1/8	27.46	876227.79	1020813.69	0/8	3.81	803953.67	1020824.57
EEG Eye State (normalized)	8/8	**-0.06**	**0.02**	**16.2**	0/8	100.2	542.0	763.41	0/8	131.31	572.73	758.52
Pla85900	5/7	0.07	**0.2**	**0.58**	2/7	**-0.02**	0.39	1.99	0/7	2.55	10.5	39.62
D15112	4/7	**0.07**	**0.15**	**0.44**	3/7	0.11	1.15	5.82	0/7	0.31	1.41	6.39
Overall Results	108/165	**0.11**	**0.61**	**4.35**	57/165	22.62	38147.66	49297.36	0/165	26.04	36793.75	49310.86

Table 4.4: Total clustering times t (in seconds) resulting from the comparison of the hybrid HPClust strategy with the competitive algorithms

Dataset	HPClust-hybrid				Forgy K-means				PBK-BDC			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	4/7	**14.71**	**27.15**	**36.39**	0/7	419.46	704.62	1696.51	3/7	60.31	76.19	105.52
HEPMASS	4/7	**6.16**	**19.99**	**28.4**	0/7	343.81	508.85	865.84	3/7	33.09	35.56	39.44
US Census Data 1990	5/7	**0.34**	**2.08**	**2.96**	0/7	29.55	61.8	120.46	2/7	4.18	4.72	5.46
Gisette	6/7	**18.01**	**21.12**	**26.19**	1/7	28.7	52.93	97.55	0/7	21.53	33.18	63.21
Music Analysis	4/7	**1.51**	**5.61**	**8.53**	0/7	49.88	86.6	145.67	3/7	5.34	7.32	10.42
Protein Homology	4/7	**1.82**	**3.37**	**5.21**	0/7	13.77	19.31	31.43	3/7	5.56	7.9	11.86
MiniBooNE Particle Identification	4/7	**2.37**	**4.33**	**6.37**	2/7	7.64	12.36	17.04	1/7	7.83	11.92	18.68
MiniBooNE Particle Identification (normalized)	4/7	**0.34**	**0.79**	**1.42**	0/7	4.07	7.14	15.28	3/7	0.93	1.21	1.77
MFCCs for Speech Emotion Recognition	3/7	**0.28**	**0.72**	**1.26**	0/7	2.99	4.91	8.07	4/7	0.67	0.94	1.3
ISOLET	0/7	1.03	3.55	4.97	0/7	1.11	1.76	3.52	7/7	**0.4**	**0.76**	**1.52**
Sensorless Drive Diagnosis	3/7	**0.78**	**1.57**	**2.71**	3/7	1.35	2.15	4.06	1/7	1.23	2.08	4.09
Sensorless Drive Diagnosis (normalized)	2/7	**0.05**	0.22	0.33	0/7	0.4	0.76	1.9	5/7	0.1	**0.15**	**0.21**
Online News Popularity	3/7	**0.18**	**0.53**	**0.87**	0/7	0.73	1.99	3.82	4/7	0.41	0.77	1.1
Gas Sensor Array Drift	0/7	0.35	1.48	2.22	0/7	0.43	0.98	2.13	7/7	**0.26**	**0.58**	**1.2**
3D Road Network	4/7	**0.15**	**0.49**	**1.28**	0/7	7.38	9.2	10.56	3/7	1.73	2.31	3.49
Skin Segmentation	1/7	**0.04**	0.15	0.21	0/7	0.17	0.3	0.64	6/7	0.06	**0.08**	**0.1**
KEGG Metabolic Relation Network (Directed)	3/7	**0.34**	**0.85**	**1.28**	0/7	1.14	1.61	2.23	4/7	1.2	1.64	2.09
Shuttle Control	0/8	0.25	0.87	1.45	3/8	**0.1**	0.19	0.41	5/8	0.11	**0.18**	**0.34**
Shuttle Control (normalized)	0/8	0.04	0.26	0.39	0/8	0.04	0.09	0.19	8/8	**0.02**	**0.02**	**0.03**
EEG Eye State	0/8	0.21	0.98	1.43	4/8	**0.07**	**0.13**	**0.22**	4/8	0.08	0.14	0.23
EEG Eye State (normalized)	0/8	0.11	0.66	0.99	2/8	**0.06**	0.14	0.33	6/8	**0.06**	**0.11**	**0.23**
Pla85900	0/7	0.11	0.93	1.47	0/7	0.13	0.26	0.58	7/7	**0.05**	**0.07**	**0.14**
D15112	0/7	0.2	0.9	1.43	0/7	0.02	0.03	0.06	7/7	**0.01**	**0.01**	**0.02**
Overall Results	54/165	**2.15**	**4.29**	**5.99**	15/165	39.7	64.27	131.67	96/165	6.31	8.17	11.85

finding favors exploring diverse K-means++ initializations to select the optimal one in the end.

The HPClust-hybrid exhibited the highest average clustering accuracy among the tested methods. This outcome was anticipated to a certain extent, as the hybrid approach combines the strengths of both regimes. In the initial stage, the competitive strategy enables extensive and rapid exploration of various K-means++ initializations on samples. In the subsequent stage, the cooperative strategy facilitates a thorough exploitation of the best solution obtained from the first stage for the remaining time. However, the hybrid strategy necessitates an additional optimization concerning the parameter T_1, which determines the split between the competitive and cooperative regimes. This parameter is highly dependent on the specific dataset and the number of clusters. In certain scenarios, particularly when dealing with numerous diverse datasets for clustering, this might pose a significant overhead that could be challenging to handle.

In examining the baseline convergence times among various parallel strategies, it was evident that the HPClust-inner method achieved quicker baseline convergence than the alternatives for the majority of datasets. This disparity was especially notable in larger datasets, as shown at the beginning of Table 4.2. For some datasets, to maintain high-quality clustering, substantial sample sizes were necessary, which were proportionate to the dataset sizes. The HPClust-inner strategy, by integrating parallelized K-means++ and K-means for each new sample, managed to expedite processing times relative to the sequential version in other parallel HPClust approaches. These findings highlight the crucial impact of algorithm selection and dataset characteristics on the delicate balance between computational efficiency and clustering accuracy. This underscores the importance of thoughtfully balancing sample size (which affects speed) with the quality of resulting clusters, as a careful trade-off is essential for achieving optimal outcomes.

Further analysis of the competitive, cooperative and hybrid HPClust strategies revealed an intricate interplay between the benefits of parallel processing and the resulting time costs. These methods did improve the solution quality, but the coordination required among mul-

tiple processors and the additional complexity from using the Numba library prolonged the convergence process, compared to the HPClust-inner method. Typically, with 8 CPUs, these strategies took up to twice as long to converge as the HPClust-inner method. This observation highlights the need to carefully weigh the trade-offs between exploiting computational resources to accelerate clustering and incurring additional overheads that may impact performance.

Table 4.3 clearly demonstrates the superiority of the HPClust-hybrid algorithm over its competitors, exhibiting a significant lead in both the number of dominant series and average overall accuracy across all datasets. The HPClust-hybrid algorithm achieves an average accuracy that is a remarkable several orders of magnitude higher than its competitors.

As shown in Table 4.4, Forgy K-means, with its linear time complexity with respect to m, predictably exhibits a significant increase in time costs for the largest datasets, exceeding the fastest HPClust version by more than 20 times. While PBK-BDC is the quickest for small datasets, its average time costs for the largest datasets are triple those of HPClust, highlighting HPClust's efficiency advantage for large datasets.

The scaling experiment results are presented in Figures 4.4a and 4.4b. For each x-axis value, the median score across 10 repetitions is displayed. The figures clearly show that all HPClust versions are highly robust and scalable with respect to the number of points, achieving optimal clustering accuracy (within 0.2% of ground truth) while keeping clustering time under 3 seconds, regardless of dataset size. In contrast, competitive algorithms Forgy K-means and PBK-BDC exhibited substantially suboptimal clustering quality, with Forgy K-means incurring unacceptable linearly rising time costs with increasing points (e.g., over 2 hours for a single execution on a 43 million point dataset). Meanwhile, PBK-BDC failed to provide steadily optimal clustering solutions at any data scale, despite slightly increased time costs for larger datasets. The HPClust versions demonstrated superior performance and scalability. Detailed experimental results, showcasing median values across various data scales and algorithms, are presented in Tables 4.5 and 4.6 for a comprehensive understanding.

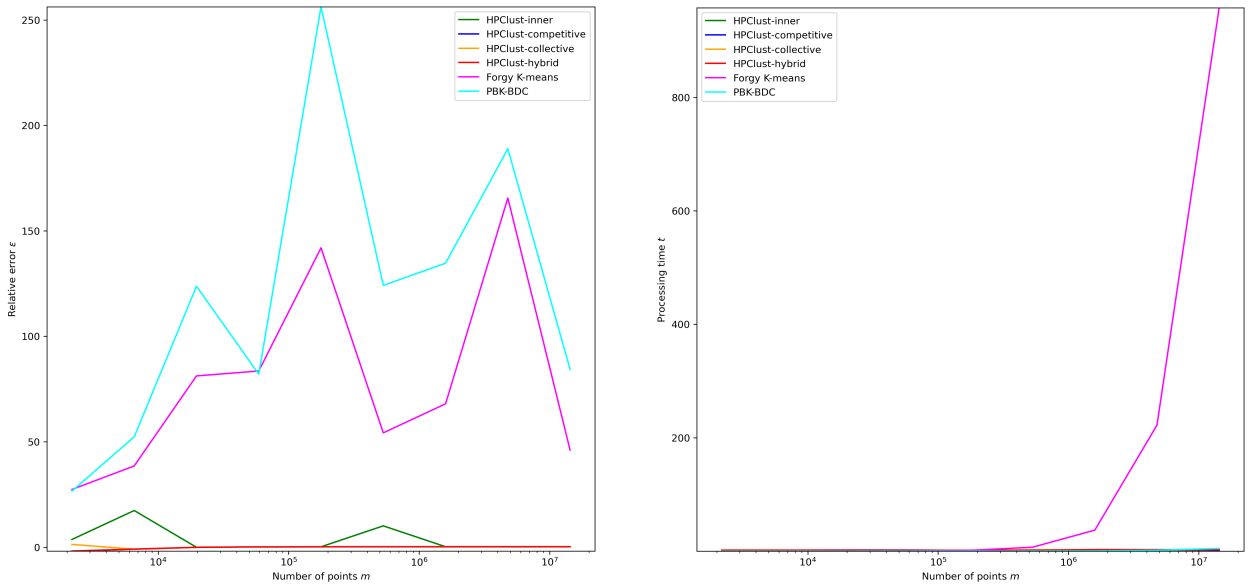
Surprisingly, the scaling experiment's results reveal an additional extraordinary property

Table 4.5: Resulting relative clustering accuracies ε for the scaling experiment in the format (*median value*, $\pm$*standard deviation*)

m	HPClust-inner	HPClust-competitive	HPClust-collective	HPClust-hybrid	Forgy K-means	PBK-BDC
3^7	3.67 ($\pm$3.90)	-1.83 ($\pm$1.54)	1.31 ($\pm$2.13)	-1.84 ($\pm$1.30)	27.38 ($\pm$16.96)	26.61 ($\pm$9.62)
3^8	17.40 ($\pm$13.71)	-0.92 ($\pm$0.01)	-0.92 ($\pm$0.01)	-0.91 ($\pm$0.01)	38.53 ($\pm$18.78)	52.35 ($\pm$22.51)
3^9	-0.04 ($\pm$18.43)	-0.06 ($\pm$0.03)	-0.06 ($\pm$0.02)	-0.05 ($\pm$0.03)	81.21 ($\pm$62.81)	123.71 ($\pm$56.46)
3^{10}	0.14 ($\pm$19.13)	0.14 ($\pm$0.04)	0.14 ($\pm$0.03)	0.17 ($\pm$0.04)	83.56 ($\pm$52.48)	82.05 ($\pm$51.69)
3^{11}	0.19 ($\pm$0.05)	0.18 ($\pm$0.05)	0.19 ($\pm$0.06)	0.19 ($\pm$0.04)	141.95 ($\pm$118.03)	256.22 ($\pm$91.90)
3^{12}	10.12 ($\pm$10.06)	0.21 ($\pm$0.05)	0.20 ($\pm$0.03)	0.20 ($\pm$0.04)	54.23 ($\pm$36.96)	124.13 ($\pm$32.12)
3^{13}	0.23 ($\pm$24.33)	0.21 ($\pm$0.03)	0.22 ($\pm$14.60)	0.20 ($\pm$0.04)	67.99 ($\pm$69.14)	134.67 ($\pm$40.73)
3^{14}	0.18 ($\pm$20.97)	0.21 ($\pm$0.03)	0.20 ($\pm$31.04)	0.22 ($\pm$0.04)	165.58 ($\pm$94.48)	188.93 ($\pm$92.47)
3^{15}	0.19 ($\pm$8.83)	0.20 ($\pm$0.02)	0.20 ($\pm$11.76)	0.22 ($\pm$0.02)	46.06 ($\pm$42.66)	84.34 ($\pm$31.01)

Table 4.6: Resulting clustering times t for the scaling experiment in the format (*median value*, $\pm$*standard deviation*)

m	HPClust-inner	HPClust-competitive	HPClust-collective	HPClust-hybrid	Forgy K-means	PBK-BDC
3^7	1.03 ($\pm$0.86)	0.62 ($\pm$0.28)	0.56 ($\pm$0.46)	1.84 ($\pm$0.59)	0.00 ($\pm$0.00)	0.00 ($\pm$0.00)
3^8	1.41 ($\pm$0.66)	1.56 ($\pm$0.75)	0.68 ($\pm$0.70)	1.82 ($\pm$0.69)	0.01 ($\pm$0.00)	0.01 ($\pm$0.00)
3^9	1.46 ($\pm$0.76)	1.99 ($\pm$0.92)	1.23 ($\pm$0.70)	1.73 ($\pm$0.86)	0.03 ($\pm$0.02)	0.01 ($\pm$0.00)
3^{10}	1.48 ($\pm$0.80)	1.96 ($\pm$0.95)	1.53 ($\pm$0.70)	1.52 ($\pm$0.78)	0.19 ($\pm$0.19)	0.03 ($\pm$0.01)
3^{11}	1.08 ($\pm$1.00)	1.41 ($\pm$0.89)	1.74 ($\pm$0.78)	1.46 ($\pm$0.72)	1.39 ($\pm$0.80)	0.06 ($\pm$0.01)
3^{12}	1.35 ($\pm$0.83)	1.72 ($\pm$0.93)	2.27 ($\pm$0.73)	1.69 ($\pm$0.64)	6.96 ($\pm$3.34)	0.18 ($\pm$0.01)
3^{13}	2.46 ($\pm$0.97)	1.47 ($\pm$0.85)	2.06 ($\pm$0.76)	2.70 ($\pm$0.81)	37.07 ($\pm$24.76)	0.53 ($\pm$0.03)
3^{14}	2.59 ($\pm$0.80)	1.48 ($\pm$0.81)	1.73 ($\pm$0.74)	2.71 ($\pm$0.94)	222.16 ($\pm$100.82)	1.62 ($\pm$0.10)
3^{15}	1.66 ($\pm$0.73)	1.64 ($\pm$0.82)	2.64 ($\pm$0.98)	2.62 ($\pm$1.21)	957.92 ($\pm$443.64)	4.81 ($\pm$0.47)



(a) Median accuracy values

(b) Median processing times

Figure 4.4: Comparative results of the algorithms with respect to the number of points m in a synthetic dataset

of HPClust: its iterative sampling processing with small samples renders it robust to noise and outliers, demonstrating a remarkable resilience to data perturbations and anomalies.

4.7.2 Trade-offs analysis

Our experiments with the HPClust algorithm have revealed several key trade-offs. Here, we present an in-depth analysis of these trade-offs, which often involve intricate balancing acts between efficiency, accuracy, computation time, and dataset characteristics. The following are the primary trade-offs that practitioners might have to consider:

1. Accuracy vs. Computation Time: Our results showed that the choice of strategy significantly influences the balance between computation time and the resulting accuracy. For example, while HPClust-inner demonstrated faster convergence times, especially for large datasets, the HPClust-competitive, HPClust-cooperative, and HPClust-hybrid

strategies offered improved clustering quality at the cost of slightly increased computation time. Thus, your choice should weigh the importance of quick results against the necessity of clustering precision;

2. Parallelism vs. Overhead: The level of parallelism used directly impacts the computation time and the overhead associated with managing multiple processors. While increasing the number of processors generally results in faster computation, it also introduces added overhead in coordinating these processors. This was particularly evident when using HPClust-competitive, HPClust-cooperative, and HPClust-hybrid strategies, which took nearly twice as long to converge as HPClust-inner, despite yielding superior solutions;

3. Sample Size vs. Quality of Clusters: The size of the sample used in the HPClust algorithm directly impacts the quality of clusters and the computation time. Larger samples often led to better approximations of the overall data distribution and improved final clustering quality. However, these benefits were offset by slower algorithmic performance, which is a crucial aspect to consider when dealing with large datasets;

4. Strategy Selection vs. Initialization Quality: In the context of HPClust, another critical trade-off lies in the choice of strategy and its influence on the quality of initializations. HPClust-competitive, which applies multiple initializations and continues clustering different K-means++ initializations to select the best one at the end, showed a slightly improved clustering quality over HPClust-cooperative. Meanwhile, the HPClust-hybrid strategy effectively amalgamated the comprehensive exploration capabilities of the competitive approach with the exploitation abilities of the cooperative approach. However, it should be noted that this comes with the requirement of additional optimization for the split parameter T_1. Therefore, the sensitivity of K-means to the quality of initial initialization is another critical factor to consider when choosing the strategy.

In navigating these trade-offs, understanding the unique requirements of your task and the nature of your dataset is paramount. Each strategy presents its own advantages and disadvantages, which should be carefully considered in light of these trade-offs. With the correct approach, these trade-offs can be effectively managed to achieve optimal clustering results with the HPClust algorithm.

4.8 Guidelines for choosing parallel strategy

Considering the outcomes of our research, we propose the following revised guidelines for selecting an appropriate parallel strategy for the HPClust algorithm:

1. If you are handling large datasets and have concerns over computation time, opt for the HPClust-inner strategy. This variant consistently showed faster convergence to baselines across most datasets, especially larger ones, as evidenced in the first rows of Table 4.2. The employment of significant sample sizes, relative to the dataset sizes, along with parallelized K-means++ and K-means on each new sample, contributed to its accelerated processing times. However, remember that larger sample sizes often led to slower algorithmic performance, so balancing sample size with the quality of clusters remains crucial;

2. When computation time is less of a constraint and you aim for better clustering quality, choose between HPClust-competitive and HPClust-cooperative strategies. Both these strategies demonstrated an improved quality of final solutions compared to other versions of HPClust, on average three times better with 8 CPUs. However, due to the additional overhead of coordinating multiple processors and the complexities associated with the Numba library, they also exhibited longer convergence times, nearly twice as long as HPClust-inner with 8 CPUs;

3. If the clustering quality is your primary focus, HPClust-hybrid or HPClust-competitive should be the preferred choices. Our findings indicated a slightly improved clustering

quality with HPClust-competitive compared to HPClust-cooperative. This improvement stems from the application of multiple initializations at the beginning, as K-means is highly sensitive to initial initialization quality. This strategy continues to cluster different K-means++ initializations, eventually selecting the best one at the end, leading to a superior solution. In the meantime, if you aim for superior clustering quality and willing to spend extra time on parameter optimization, opt for the HPClust-hybrid strategy. This choice demonstrated the best resulting clustering quality, while retaining the same degree of time efficiency as the competitive and cooperative approaches.

These guidelines should assist researchers and practitioners in choosing an appropriate parallel strategy for their specific needs. However, keep in mind that these are general guidelines, and the choice of parallel strategy should be adapted to the specific requirements of your task and the nature of your dataset. This research strongly suggests that parallelism, when feasible, offers a significant enhancement in clustering accuracy and convergence time compared to the sequential variant.

Overall, the best strategy is likely to be one that strikes a balance between the need for accuracy, computation time, and the specific characteristics of the dataset at hand. The effectiveness of each strategy will inevitably depend on these factors, and the choice should be made accordingly.

4.9 Conclusion and future research

This chapter introduces the HPClust algorithm and explores its four parallel strategies on diverse datasets, including real-world and synthetic ones. Our comprehensive evaluation focuses on three essential metrics: relative clustering accuracy (ε), total runtime (t), and baseline-normalized runtime ($\bar{t}$). These metrics provide a thorough assessment of each strategy's effectiveness and efficiency, enabling a well-rounded comparison.

The experimental results demonstrate HPClust's unrivaled effectiveness, efficiency, and scalability compared to baseline algorithms across a vast range of real-world datasets (span-

ning small to big sizes) and synthetic datasets. HPClust consistently outperforms its competitors, showcasing remarkable robustness to data scale and noise, as well as adaptability in various data settings.

Also, this research demonstrates that no single parallel strategy universally optimizes the HPClust algorithm. Instead, the most effective approach depends on the dataset's characteristics, emphasizing the need for adaptive techniques that dynamically select the best strategy. However, in most cases, we recommend practitioners to employ either the competitive or hybrid (competitive-cooperative) parallel strategies of HPClust, which have shown superior performance and versatility.

Additionally, our work offers a comprehensive review of the primary high-performance techniques utilized for optimizing data clustering algorithms. We delve into the intricate aspects and nuances of applying parallel techniques, specifically analyzing the challenges and pitfalls associated with the HPClust algorithm. Through a detailed trade-off analysis, we provide practical guidelines to assist in selecting the most suitable parallel strategy for specific use cases. These guidelines aim to facilitate informed decision-making and provide actionable recommendations.

Future research will focus on developing adaptive methods that can intelligently choose the most suitable parallel strategy based on the specific dataset, optimizing performance and accuracy. Additionally, we will conduct a more in-depth analysis of the trade-offs revealed in this study, exploring their nuanced effects on algorithmic performance and accuracy, to uncover actionable insights for further improvement.

Another promising future research direction for the proposed HPClust algorithm is its potential adaptation for clustering streaming datasets or continuously growing datasets. This is particularly relevant in scenarios involving IoT sensors, financial transactions, social media feeds, and other real-time data sources, where data is constantly generated and requires efficient clustering techniques to uncover insights and patterns. By extending HPClust to handle streaming data, researchers can unlock new opportunities for real-time analytics and decision-making in various fields.

This study's findings and observations lay the groundwork for advancing efficient and adaptive parallel techniques for HPClust and beyond. Our goal is for this research to make a meaningful impact in the fields of data clustering and high-performance computing, driving innovation and improvement in these areas. By shedding light on the complex relationships between parallel strategies, dataset characteristics, and algorithmic performance, we aim to spark further discovery and progress.

Chapter 5

COMPARATIVE ANALYSIS OF OPTIMIZATION STRATEGIES FOR K-MEANS CLUSTERING IN BIG DATA CONTEXTS: A REVIEW

5.1 Introduction

5.1.1 Motivation and novelty

In today's landscape, selecting the right clustering paradigm and tool for new big datasets is not a straightforward endeavor for practitioners. Without proper guidance, comprehending the strengths and constraints of existing clustering models within big data scenarios becomes challenging. Additionally, gaining insights into the range of appropriate big clustering algorithms tailored for the specific dataset can prove to be daunting.

The minimum sum-of-squares clustering (MSSC) is a more formal and scientific name for the optimization problem (1.1), meaning all methods related to MSSC that are not necessarily similar to K-means. However, within the broad literature, MSSC is also often referred to as the K-means clustering problem due to the popularity, simplicity, and efficiency of the K-means algorithm in solving it. Our primary interest lies in various enhancements and modifications of the K-means algorithm designed to optimize its application to large datasets, which we call "K-means-like methods". This chapter specifically focuses on these advanced and optimized techniques for improving the standard K-means. While these may include new methods, they must either share some similarities with the standard K-means or employ it within a higher-level heuristic as a form of local search. A review of the methods that are entirely unrelated to K-means yet still address the minimum sum-of-squares clustering (MSSC) problem is of secondary interest to us.

Our work in this chapter aims to closely examine various K-means-like methods proposed

in existing literature to tackle the complex challenges of clustering big datasets. We will conduct a thorough survey of these methods, categorizing them into distinct groups. For each category, we will explore the latest advanced algorithms they encompass. Moreover, we will closely evaluate the strengths and limitations inherent in each approach. Following the experimental setup of Chapter 3, a comprehensive experimental analysis will justify our findings. In our analysis, we adopt the LIMA dominance relation [Bri+23] (thoroughly discussed in Section 2.7) as the principal metric for comparing various big data clustering algorithms. To the best of our knowledge, this study signifies the innovative use of this criterion for evaluating big data clustering algorithms.

The LIMA dominance criterion considers all three essential properties of an algorithm: final accuracy, processing time, and simplicity. The latter aspect becomes critical in the context of big data. We have undertaken an extensive evaluation of the most cutting-edge K-means-like clustering algorithms for big data, available from the recent literature, using a large array of real-world datasets. Also, this chapter for the first time proposes practical recommendations and a flowchart intended to assist in selecting clustering algorithms for big data.

Contrary to the experiments presented in Chapter 3, where Big-means was benchmarked against basic K-means variants and a few advanced state-of-the-art hybrid heuristics, the studies of the current chapter predominantly concentrate on conducting an exhaustive survey and comparison (analytical and experimental) of a diverse set of existing approaches for optimizing K-means-like algorithms in the context of big data.

Thus, the primary aim of this chapter is to offer a comprehensive review of the techniques for optimizing K-means and, secondly, other MSSC-related clustering approaches in big data environments. We reinforce our findings with extensive experiments and interpret the results using the LIMA dominance criterion. In doing so, we aspire to make a novel and significant contribution to the progress of research in the dynamic field of big data clustering.

5.1.2 Chapter structure

The chapter is structured as follows. Section 5.2 reviews diverse challenges associated with big data processing. Section 5.3 gives an in-depth analysis of various techniques for optimizing K-means clustering and other clustering approaches related to MSSC in big data environments, exhibiting their notable representative algorithms. Section 5.4 discusses additional information drawn from other available review papers. Based on the conducted literature analysis, Sections 5.5 and 5.6 offer generalizing thoughts along with useful practical advices. Section 5.7 is devoted to the conducted experiments and the analysis of their results. Finally, Section 5.8 concludes the chapter and outlines future research ideas.

5.2 Big data challenges

Big data clustering is a complex task that faces several challenges that can be grouped into a few categories. One of these categories is related to data handling and manipulation, including data preprocessing [AB17], noise and outliers [LG21], storage and retrieval [APK22], and imbalanced data [WWG20]. These difficulties often arise from the complex and irregular nature of big data.

Another category is concerned with the computational and practical aspects of clustering algorithms, such as scalability [KH14], computational complexity [MHE21b], simplicity [MM19], as well as ease of implementation and parallelization [AAS21].

The selection and application of clustering algorithms form the third category. Choosing an appropriate clustering algorithm and its parameters is crucial for effective data analysis, as different data types may require unique clustering methods. With the variety present in big data, it is particularly important to choose suitable clustering techniques to ensure successful insights.

The fourth category involves specific computational requirements and limitations, such as computational resources [SJ16], time-dependency, real-time analysis [HS16], data sparsity [ŚHT19], high-dimensional data [WKK15], and multi-view data [ZZZ17].

Lastly, reproducibility and transparency are essential challenges that need to be addressed to ensure the validity and reliability of the results.

In the following paragraphs, we will discuss each challenge in more detail.

Data preprocessing is often a critical initial step to ensure the best possible clustering results [AB17]. Preprocessing techniques such as removing noise, reducing dimensionality, and handling missing data may be required to improve the quality of the data. However, preprocessing can be time-consuming, especially when working with large datasets.

Big datasets often contain a large amount of irrelevant information and data points that are significantly different from the rest, known as noise and outliers. These can negatively impact the clustering results and create clusters that are not meaningful, making it difficult to identify real clusters. Therefore, it is crucial to employ an efficient clustering algorithm that can handle this situation to obtain accurate clustering results [LG21].

Storing and retrieving large datasets can present a challenge, particularly when real-time or near-real-time analysis is required. Big data clustering may necessitate specialized distributed storage and retrieval systems capable of handling large data volumes and providing quick access to data for analysis [APK22]. Furthermore, some big datasets may be too large to fit into the RAM of a computing system. Also, data imbalance often occurs in big datasets, where some classes or clusters have significantly more data points than others. This can cause clustering algorithms to prioritize larger clusters and disregard smaller ones [WWG20].

Scalability can be defined as the ability to find the balance between the quality of the obtained solution and the amount of processed information (computational cost) in response to changes in volume and configuration of input data [Mus+23]. Traditional clustering algorithms that work well with small and large datasets may not be suitable for big data due to its increased computational complexity and memory requirements. Additionally, big data is often stored across multiple machines and storage systems, making it challenging to perform clustering analysis efficiently. To handle big data, clustering algorithms need to be designed to utilize distributed computing architectures and parallel processing [KH14].

The computational complexity of traditional clustering algorithms can make analyzing

big datasets difficult within a reasonable amount of time. This is especially true when there is a need for real-time or near-real-time analysis. As noted by Mahdi et al. [MHE21b], the time required to perform clustering on large datasets can be prohibitive.

Modern clustering algorithms often prioritize complexity over simplicity, leading to increased time complexity and challenges in implementation across various computing architectures, including parallel systems [MM19]. The number and types of instructions used in these algorithms contribute to their complexity. This complexity, while potentially enhancing accuracy, tends to limit their practical use, especially for non-experts. The research trend towards these hybrid and complex algorithms might be a misleading scientific direction. Instead, there is a growing need to focus on developing simple yet effective algorithms that ensure wider applicability and accessibility in practical scenarios.

Parallelizability is essential in the context of clustering algorithms as it refers to an algorithm's capability to decompose the initial, large task into multiple smaller subtasks. This decomposition enables the concurrent (parallel) and independent processing of these subtasks by several processors or workers, shortening the overall processing time. The final solution to the original task is then derived by integrating the solutions of these smaller subtasks. Parallelization enhances an algorithm's efficiency and scalability by effectively decomposing tasks and enabling parallel processing, while also optimizing the use of available computing resources and speeding up data processing. As noted by Alguliyev et al. [AAS21], this is a crucial consideration for any clustering algorithm, particularly those designed to handle big datasets.

Choosing the right clustering algorithm and its parameters is essential for each dataset. There are various clustering algorithms available, each with its own unique strengths and weaknesses. However, selecting the most suitable algorithm for a specific dataset can be challenging, particularly when dealing with large datasets. Different algorithms may have varying computational requirements, and some may not be scalable to large datasets. Furthermore, most advanced clustering algorithms involve a large number of hyperparameters but do not provide clear guidelines for their optimal selection. Performing an ordinary grid

search may be unfeasible in big data conditions due to unacceptable time costs. Also, despite the usually better accuracy obtained by hybrid clustering algorithms, the necessity of optimizing a large number of hyperparameters is a negative factor that impedes their widespread adoption.

Clustering big data can require substantial computational resources, including processing power, memory, and storage [SJ16]. Thus, to perform clustering analysis on big data, organizations may need to employ high-performance computing (HPC) technologies, including investing in specialized hardware or cloud computing services.

Some big datasets may have a time-dependent nature, so the patterns and relationships in the data may change over time [HS16]. Clustering algorithms may need to be adjusted to handle the time-dependency of the data and ensure that the clustering results remain relevant. Real-time or near-real-time clustering of big data can be challenging because of the need for fast processing and the large volume of data. Developing real-time clustering algorithms that can handle big data is an active area of research [HS16]. Also, big datasets can be sparse, meaning that most data points are zero or missing. Sparse data can lead to problems with clustering algorithms as the distance between points becomes less meaningful [ŚHT19]. Additionally, dealing with big data can become challenging due to the high number of dimensions that the data can have, making it difficult to identify natural clusters and visualizing the data. This can lead to the "curse of dimensionality" where the distance between points becomes less meaningful as the number of dimensions increases [WKK15].

Multi-view data, also known as multi-source data or multi-modal data, refers to a type of data that has multiple perspectives (views) or modalities, each representing a different aspect of the same object or phenomenon. Combining all the views provides a more accurate representation of the data. For example, in a multimedia database, a video clip can be represented by multiple views, such as visual information, audio information, and textual information, each representing a different aspect of the video. Big data usually comprises large and complex datasets that include multiple modes or sources of information, such as text, images, videos, audio, and other types of data [MHE21b]. Multi-modal data can be

non-uniform, meaning that the distribution of data points across the different modes may not be equal. Clustering algorithms, which typically use distance measures to determine the similarity between data points, can find it challenging to identify appropriate distance measures for each modality [ZZZ17].

In scientific research, reproducibility and transparency are essential components. However, when dealing with complex algorithms, clustering results on big data can be challenging to replicate. To ensure reproducibility and transparency, it is necessary to document the clustering process carefully and share the code and data.

To summarize, all the above points show that the emergence of big data entails a set of unique challenges. To extract insights and value from vast datasets, there is a need for specialized clustering techniques ("true big data" algorithms). These algorithms are specifically designed to address the big data issues and the inadequacy of classic approaches. Nowadays, this is an active area of research. In this chapter, we will provide a comprehensive review of the numerous attempts made in the literature to create big data clustering algorithms.

5.3 K-means optimization approaches

The literature proposes numerous techniques to optimize K-means and address the challenges of MSSC in big data contexts, as discussed above. In this section, we will review these techniques and classify them into separate categories. For each category, we will consider its corresponding state-of-the-art algorithms. Also, we will analyze the strengths and weaknesses of each approach and conduct an extensive experimental analysis. The objective of this study is to offer a thorough review of the techniques used for optimizing K-means clustering and other clustering approaches related to MSSC in big data environments, as the MSSC concept is often used interchangeably with K-means clustering.

Figure 5.1 shows an ontological graph that illustrates the problem area and its main technologies used in optimizing K-means for big data. It shows the relationships between different concepts and technologies, and how they contribute to the overall field.

Figure 5.2 shows the frequency of different big data clustering approaches or technologies

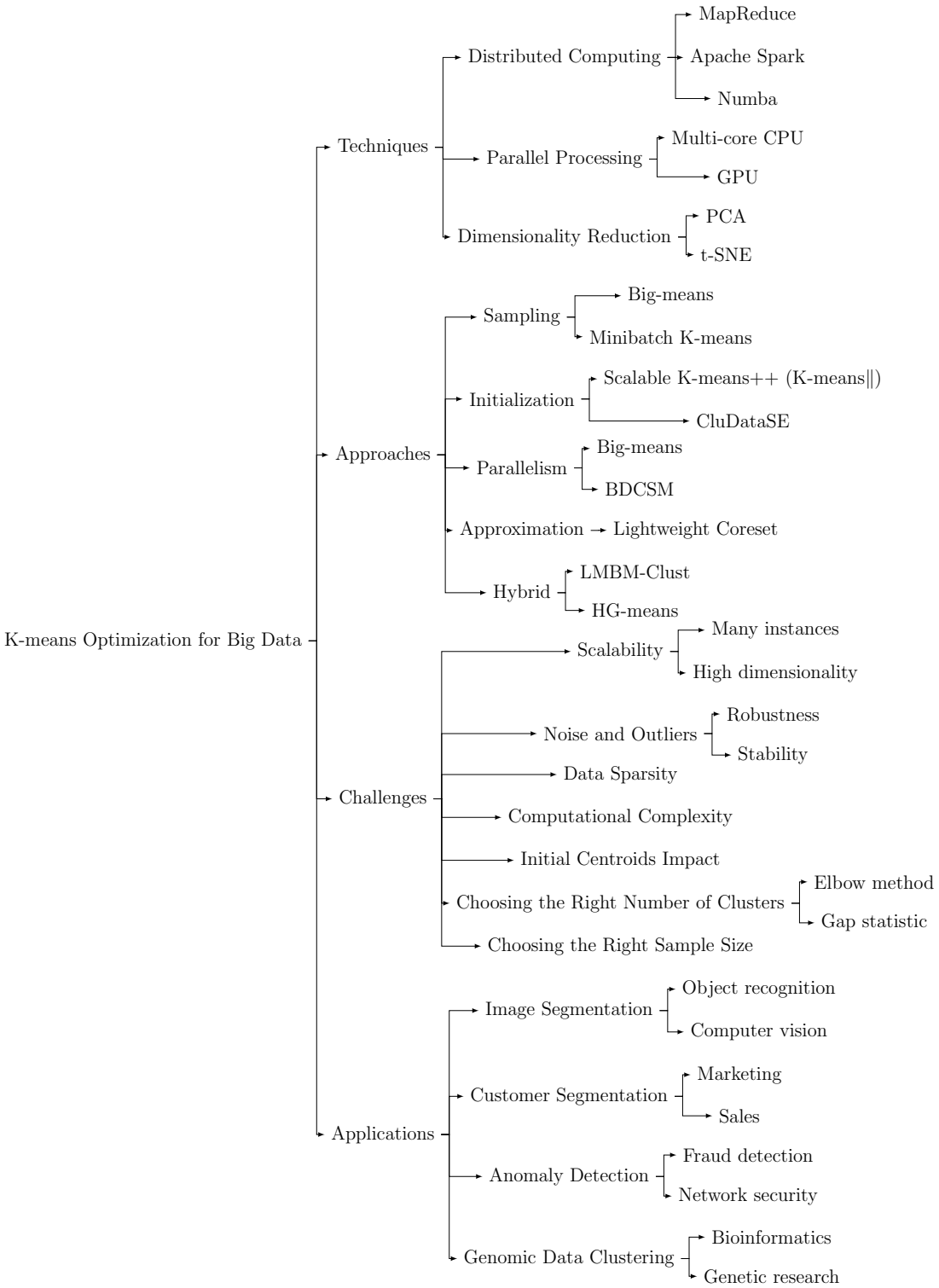


Figure 5.1: Ontological graph of the problem area and its main technologies

used in the literature. It can help readers understand which methods are most commonly used or discussed in the field. Each number in the histogram signifies the number of published papers with the given keyword contained in the title or the abstract. The statistics was obtained by searching the Web of Science Core Collection by keywords.

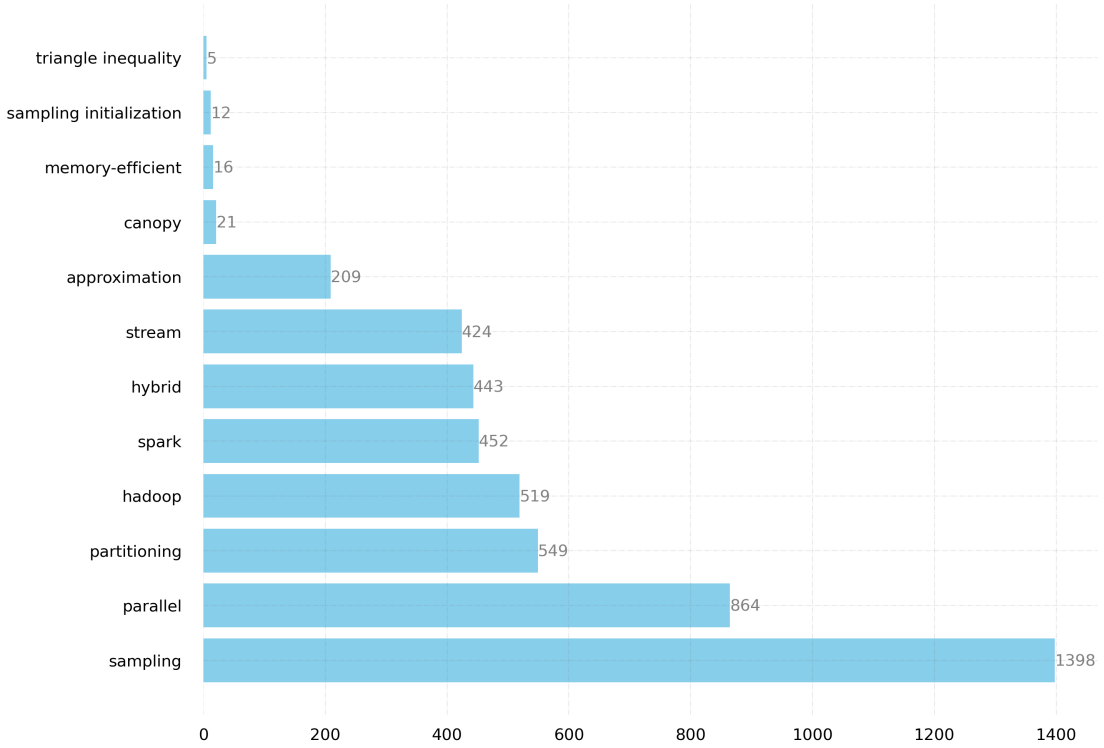


Figure 5.2: Histogram of most commonly used approaches and technologies in the field of big data clustering

Figure 5.3 shows a timeline of key developments in K-means clustering optimization and other MSSC algorithms.

In subsequent subsections, we provide a detailed examination of the ontological graph branch associated with big data clustering techniques.

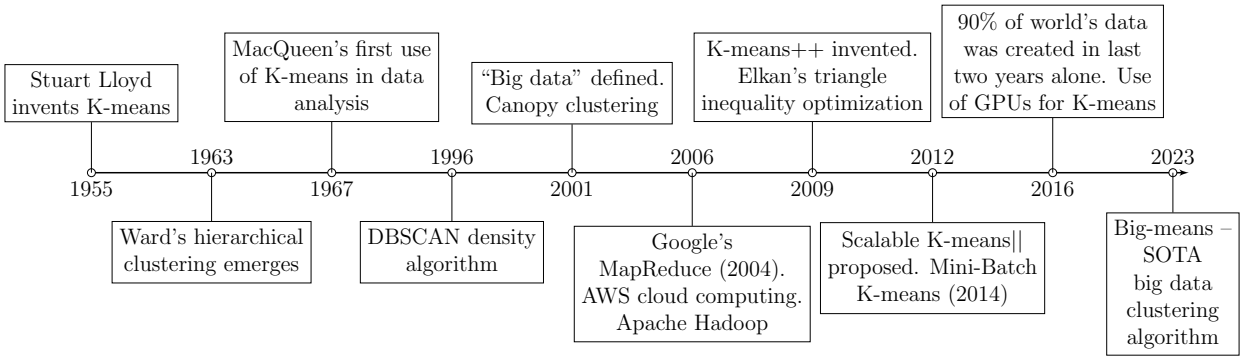


Figure 5.3: Timeline of milestones in K-means clustering optimization and other MSSC algorithms

5.3.1 Decomposition

Decomposition is a common problem solving technique. It splits the initial problem into a set of subproblems whose aggregate difficulty does not exceed the difficulty of the initial problem. By aggregating the solutions from the subproblems, it is possible to create a solution for the whole original problem.

Decomposition of the MSSC problem is such a set $\mathbb{D}$ consisting of q subsets X_i of X, each of which has the same cardinality $s \in \mathbb{N}$:

$$\mathbb{D} = \{X_i,\ i \in \mathbb{N}_q :\ X_i \subset X,\ |X_i| = s\}\,. \tag{5.1}$$

After solving the subproblems defined on X_i for $i \in \mathbb{N}_q$, which can be performed in parallel, the obtained local solutions are aggregated in some way to form a solution for the original problem defined on X. This approach is an instance of the more general divide-and-conquer paradigm. Put simply, this paradigm involves two primary phases in solving an optimization task: decomposition and aggregation. Decomposition of a large dataset into relatively small samples and their subsequent parallel processing using multiple processors is a practical tool for improving scalability [KMM20].

Decomposition can be performed in different ways:

(a) Partitioning. This can be achieved using techniques such as vertical partitioning or horizontal partitioning. In vertical partitioning, the data is partitioned based on the features, whereas in horizontal partitioning, the data is partitioned based on the instances. This approach can be effective when the data is too large to fit into memory or when the computation time is a bottleneck;

(b) Sampling. This approach involves selecting a random subset of the data and clustering it. The resulting clusters can then be used as an approximation of the clusters that would be obtained from clustering the entire dataset. This approach can be effective when the data is uniformly distributed and the clusters are well separated;

(c) Stream-based clustering. This approach is also called the online or incremental clustering. It involves clustering data points in small batches in real-time as they arrive, without needing to store the entire dataset in memory. Thus, clusters are updated as new batches arrive. This approach can be effective when the data is continuously streaming, and real-time clustering is required (the clusters need to be updated dynamically).

All three decomposition methods are tightly interrelated. Each method can be naturally transformed into another one. In Chapter 3, we proposed a new state-of-the-art clustering algorithm for big data, Big-means. Due to its key properties, Big-means is able to combine all three approaches to decomposition.

Minibatch K-means [Scu10] is a basic algorithm that utilizes the sampling approach. In each iteration of the MiniBatch K-means algorithm, a minibatch (a random subset of the data) is selected for processing. Then this minibatch goes through the assignment and update steps, as in the standard K-means. However, rather than updating each centroid by averaging all data points assigned to it, the update is made using a fraction of the points in the minibatch. Specifically, the new centroid position is a weighted average between its old position and the mean of the new points assigned to it. The weights ensure more recent

data has a larger influence on the centroid position, allowing the algorithm to better adapt to changes in the data distribution.

The algorithm continues to iterate, selecting new minibatches and repeating the assignment and update steps, until a stopping criterion is met. This could be a fixed number of iterations, or a point where the centroids no longer move significantly between iterations.

Thus, the Minibatch K-means algorithm consists of the following key steps:

1. Initialization. Randomly select k points from the dataset to act as the initial cluster centroids;

2. Sampling. Draw a random minibatch M of size s from the dataset;

3. Assignment. Assign each data point in the minibatch to its nearest centroid;

4. Update. Update the centroids by calculating the mean of all data points assigned to each centroid;

5. Repeat. Repeat steps 2 – 4 until the centroids no longer move significantly.

The pseudocode for the Minibatch K-means algorithm is showed in Algorithm 11.

In each iteration of Minibatch K-means, the centroids are updated only using the points of an incoming minibatch, making the algorithm more memory-efficient and scalable than traditional K-means. However, the quality of the solution may be drastically affected by the randomness of the minibatch selection, and lots of random initializations may be required to obtain an optimal result. In Chapter 3, the Big-means algorithm [Mus+23] was proposed, which addresses these limitations by implementing a stable and effective K-means++ initialization for the initial sampled subset and degenerate clusters. It also introduces an advanced centroid update criterion based on the optimal objective function value obtained from all processed samples. Furthermore, Big-means deepens the exploration of each new sample by employing a K-means local search strategy. All these modifications significantly enhance the stability and precision of the clustering outcomes [Mus+23].

Algorithm 11: Minibatch K-means Clustering

Function `Minibatch_k_means`(X, k, s, T)**:**
  Input : Feature vectors $X = \{x_1, \ldots, x_m\}$;
    Desired number of clusters k;
    Minibatch size s;
    Number of iterations T;
  Output: Centroids $C = \{c_1, \ldots, c_k\}$;
    Point-to-cluster assignments $Y = \{y_1, \ldots, y_m\}$.
  Initialize all k centroids C randomly on X;
  Set iteration counter $t \leftarrow 0$;
  while $t < T$ **do**
    Draw a random minibatch M of size s from X;
    for *each point* x *in* M **do**
      Assign x to the nearest centroid in C;
    end
    for *each centroid* c *in* C **do**
      Update c by taking the mean of all points assigned to c;
    end
    $t \leftarrow t + 1$;
  end
  $Y \leftarrow$ Assign each point in X to nearest centroid in C;
  return $C, \quad Y$

The Big-means algorithm represents a synthesis of three key approaches to data decomposition. This simple method begins by sampling a subset S from the dataset X, containing s data points, a small fraction of the total m. It leverages the K-means++ algorithm for the initial selection of centroids C from S. Subsequently, it employs a dynamic clustering process, utilizing K-means with the most effective centroid set found across previous samples. This iterative refinement, guided by the "keep the best" strategy, continually updates to the incumbent solution, based on the sum-of-squares criterion evaluated on current samples.

Unique to Big-means is its strategy for handling empty clusters: it rejuvenates these by reapplying K-means++ to produce new potential centers, thus avoiding the pitfalls of traditional methods and enhancing the solution's quality by broadening the search for minimizing the objective function. Another pivotal feature of Big-means is the "shaking procedure", which generates new samples to perturb the current centroid solution, thereby infusing diversity and adaptability into the clustering process. By treating the dataset as a point cloud in the n-dimensional space, each sample acts as a sparse representation, injecting fresh perspectives into each iteration.

Upon reaching a predetermined time limit or processed sample count, Big-means assigns all data points to the best obtained centroid configuration C. However, this step can be either omitted or adjusted according to the specific requirements of an application area. The algorithm's iteration time complexity stands at $\mathcal{O}(s \cdot n \cdot k)$, offering a significant speed advantage for big datasets, especially when compared to the conventional K-means or K-means++ algorithms.

Big-means is designed to marry the swift execution of K-means with high-quality clustering outcomes through intelligent successive refinements across random data subsets, adeptly managing empty clusters. This iterative, sample-based approach not only ensures scalability for big data but also mitigates the risk of settling into suboptimal solutions by continually refreshing the search space and centroids. The algorithm's stochastic optimization approach, through iterative random sampling, ventures into diverse objective function landscapes, enhancing the likelihood of discovering superior configurations than possible with a single

K-means execution. Big-means exemplifies a blend of deterministic and probabilistic strategies to achieve a balance between exploration and exploitation in the solution space. This balance is crucial for the algorithm's ability to navigate through and beyond local optima, towards a globally optimal clustering arrangement.

The research in Chapter 3 benchmarks Big-means against established clustering algorithms, revealing its superior performance in accuracy and speed across various dataset sizes, thereby setting a new standard for "true big data" clustering methodologies. Algorithm 5 encapsulates the Big-means algorithm.

By iterating through samples and refining centroid positions, Big-means efficiently addresses the computational complexity challenge by reducing computational demands while preventing stagnation at inferior solutions, a common issue inherent to traditional approaches. By treating each sample as an incoming time-dependent data chunk, Big-means can also tackle the challenge of real-time analysis in big data. Big-means can accept non-overlapping samples, facilitating the distribution of input data across separate computing nodes, thereby addressing the challenge of data storage and retrieval within big data environments. Importantly, these advantages of Big-means do not increase the algorithm's time complexity, nor do they compromise the simplicity of its implementation.

Alguliyev et al. took a different approach in their Big Data Clustering on a Single Machine (BDCSM) algorithm in [AAS21]. BDCSM partitions large datasets into chunks, clusters them in parallel using K-means, and aggregates the resulting cluster centers into a final pool. Then, the algorithm clusters the pool using K-means. The pseudocode for the BDCSM algorithm is presented in Algorithm 12. BDCSM is a prominent instance of partition-based parallel clustering algorithms. While BDCSM is a straightforward algorithm that leverages parallel processing to accelerate clustering and partially addresses the challenge of distributed storage for big data, it fails to address the initialization issue, resulting in reduced clustering accuracy (effectiveness) despite its gains in efficiency.

The empirical evaluation of BDCSM in the original paper showed its superiority over the classical K-means algorithm [AAS21]. However, this evaluation did not demonstrate any

superiority of BDCSM to other more advanced algorithms for clustering large datasets.

Algorithm 12: BDCSM Clustering

Function `BDCSM(`X, p`)`**:**
  Input : Feature vectors $X = \{x_1, \ldots, x_m\}$;
    Chunk size p;
  Output: Centroids $C_{\text{final}} = \{c_1, \ldots, c_k\}$;
    Point-to-cluster assignments $Y = \{y_1, \ldots, y_m\}$.
  Partition the dataset X into chunks of size p;
  for *each chunk* C_i **do**
    Cluster C_i using randomly initialized K-means to obtain centroids $C_{i,\text{new}}$;
    Add $C_{i,\text{new}}$ to the pool of centroids P;
  end
  Cluster the pool P using randomly initialized K-means to obtain final centroids C_{final};
  $Y \leftarrow$ Assign each point in X to nearest centroid in C_{final};
  return $C_{\text{final}}, \quad Y$

Notable examples of online clustering algorithms include online K-means [CGa21] and online spectral clustering [Nin+10]. The pseudocode for the online K-means clustering algorithm is provided in Algorithm 13. Online K-means updates the cluster centroids incrementally as each new data point arrives, without the need to access the entire dataset. This gives the Online K-means algorithm an advantage over traditional K-means in terms of processing real-time data and scalability to large datasets. However, it still shares several limitations with Minibatch K-means and BDCSM, including centroid initialization issues, convergence to local optima, and ignoring degenerate clusters. Furthermore, Online K-means cannot process entire chunks of incoming real-time data, making the final clustering outcome highly sensitive to the sequence of incoming data points.

With sufficient computational resources, data decomposition enables the application of advanced parallel and distributed techniques to individual subproblems, as explored in the next subsection, unlocking vast opportunities for more efficient processing and analysis.

Algorithm 13: Online K-means Clustering

Function `Online_k_means(`X`)`**:**
  Input : Streaming dataset $X = \{x_1, x_2, \ldots\}$;
    Number of centroids k (optional, for initialization);
  Output: Centroids $C = \{c_1, \ldots, c_k\}$;
    Point-to-cluster assignments $Y = \{y_1, y_2, \ldots\}$.
  Initialize C with k centroids chosen from the first batch of X;
  while *new data point* x *arrives* **do**
    Find the nearest centroid c_i in C to x;
    Update c_i by incorporating x into the calculation: $c_i = (c_i \cdot n_i + x)/(n_i + 1)$;
    Increment the count n_i for centroid c_i;
  end
  Optionally, periodically re-evaluate and adjust centroids with a batch process;
  $Y \leftarrow$ Assign each point in X to the nearest centroid in C;
  return C, Y

5.3.2 Parallelization and distributed computing

Parallelization and distributed computing are two common techniques for optimizing K-means-like methods in big data settings [AAS21]. Most commonly, parallelization involves breaking the data into smaller subsets and clustering them simultaneously on multiple processors. The results are then combined to generate the final clusters. This approach can be useful when the data is too large to fit into memory or when the computation time is a bottleneck. Alternatively, big data can be distributed across multiple machines. In this case, clustering is performed in a distributed manner, using frameworks such as Apache Hadoop or Apache Spark. This approach can be effective when the data is too large to fit on a single machine. By distributing the data and computations, the workload can be divided among multiple machines, reducing the processing time and allowing for scalability.

Apart from the BDCSM algorithm discussed above, paper [BBE19] proposes the "Sampling, Triangle inequality and MapReduce" (STiMR) K-means algorithm, which leverages the MapReduce framework at every stage of the clustering process. STiMR represents a typical approach in the class of parallel and distributed clustering algorithms, employing

the *map* and *reduce* operations on groups of input data points. Specifically, it leverages MapReduce to draw a random sample from the big dataset using reservoir sampling, cluster the sample using K-means with the triangle inequality, and assign the data in the original dataset to the obtained centroids. Technically, the STiMR K-means algorithm is an accelerated version of applying K-means to a single random sample, with the added benefit of MapReduce parallelization at every stage.

However, the logic of some other big data clustering algorithms allows for other types of parallelization. For instance, independent or somewhat dependent loops inside a clustering algorithm can be naturally parallelized. Big-means algorithms [Mus+23] is a good example that is amenable to this kind of parallelization. Specifically, each execution of the loop at line 4 in Algorithm 5 can be assigned to a separate parallel worker. These workers can be totally independent from one another (competitive parallelization) or share information about the best sample solutions (collective parallelization). The former mode promotes exploration among the initial solutions, while the latter ensures a thorough exploitation of the best initial solution. Additionally, one can consider a hybrid parallelization mode, in which for some number of iterations the competitive mode is employed until being switched to the collective mode. The hybrid mode is expected to combine both benefits of the competitive and collective parallelization schemes, achieving an optimal trade-off between them.

Throughout our experimental phase, we leveraged the Numba library for Python to parallelize only those specific parts of the big data clustering algorithms that were explicitly highlighted as parallelizable within their original research papers. Numba [LPS15; Mar18] is a key instrument in high-performance computing, featuring optimization capabilities such as parallelization, multi-threading, and vectorization. These features are core strategies in performance optimization, transforming the execution speed of Python functions, loops, and numerical computations. Numba's dynamic generation of optimized machine code for both CPUs and GPUs further contributes to this performance boost, converging Python's usability and the speed of lower-level languages.

While the MapReduce framework may offer advantages in terms of scalability and fault

tolerance, particularly for very large datasets, we decided not to use it in our experiments. Our goal was to ensure that all algorithms being tested were parallelized under equal conditions, in order to enable fair and accurate comparison of their performance in terms of parallelizability. Thus, we parallelized all algorithms using the Numba library, which allowed us to control and optimize the parallelization process for each individual algorithm.

To maximize memory efficiency in MSSC clustering, the next subsection describes how algorithms may employ more frugal data structures to represent the dataset, which are then either directly passed to K-means for iterative refinement or in some other way adapted to yield an MSSC solution.

5.3.3 Memory-efficient clustering

Memory-efficient clustering algorithms are designed to handle datasets that are too large to fit into memory. These algorithms typically take advantage of some form of data summarization or use disk-based or online methods to manage memory usage. Often, memory-efficient data structures are employed to reduce the memory footprint. Memory-efficient clustering algorithms can be effective when the memory available is limited. They can be incredibly useful for big data applications where datasets may consist of millions or billions of data points, far beyond what traditional clustering algorithms can handle.

Here are a few examples of memory-efficient clustering algorithms and techniques:

1. BIRCH [ZRL96] (Balanced Iterative Reducing and Clustering using Hierarchies): BIRCH is an efficient clustering method designed for large datasets that may not fit into memory. It creates a memory-efficient, hierarchical data structure called a CF (Clustering Feature) Tree, which holds summarized information about the data, rather than storing all data points. The CF Tree is built in a single pass through the data, where each data point is either integrated into an existing cluster or used to create a new one, based on a distance threshold. By storing and manipulating summaries of data points, rather than the entire dataset, BIRCH is able to perform clustering operations more

efficiently, making it particularly well-suited for handling big data. BIRCH can indirectly solve the MSSC problem by using the centroids of the micro-clusters formed by the leaf nodes of its CF Tree as initial seeds for the K-means algorithm, which refines these centroids to minimize the within-cluster sum of squared distances. This two-step approach combines BIRCH's efficiency in handling large datasets with the precision of MSSC algorithms in optimizing cluster centroids;

2. DBSCAN [Est+96] with Incremental Updates: Variations of the popular DBSCAN algorithm have been developed that incrementally update the clustering structure with the arrival of new data points, reducing the memory required to store the entire dataset at once. Thus, these algorithms can apply a density-based clustering to portions of a dataset one at a time, without the need to process the whole dataset at once. This version of DBSCAN can be adapted to the MSSC paradigm by using it as preprocessing step on each newly arriving data portion, and passing the obtained centroids as initialization to the K-means clustering on the data portion;

3. CluStream [Agg+03]: The CluStream algorithm effectively clusters evolving data streams by creating and updating lightweight microclusters in real-time with each incoming data point. These microclusters, each summarized by a "Clustering Feature" vector, dynamically evolve, allowing for the creation of new microclusters, deletion of outdated ones, or merging of close clusters, providing an adaptive response to changes in data distribution. Periodically, these microclusters are consolidated into macroclusters using the K-means algorithm, which offer a longer-term, comprehensive view of the data stream's overall clustering structure;

4. Canopy Clustering [MNU00]: The Canopy Clustering algorithm is a pre-clustering technique used to speed up other clustering algorithms by reducing the size of the dataset. The algorithm works by scanning through the data and creating canopies (overlapping groups of data points) using cheap distance measures, which form a first

approximation of the true clusters. These canopies are then used to partition the data, reducing the computational cost for a subsequent run of a more expensive clustering algorithm like K-means or hierarchical clustering. Canopy clustering is often used as a preprocessing step for the K-means algorithm or a hierarchical clustering algorithm.

By employing memory-efficient algorithms, it becomes possible to perform clustering on big datasets. However, these memory-efficient techniques often require careful tuning and consideration of their parameters, and might make assumptions or approximations that could limit their effectiveness depending on the specific application.

Several memory-efficient hierarchical clustering algorithms have been developed that are scalable to handle large datasets. Although these algorithms do not employ K-means and cannot be classified as MSSC algorithms in a direct sense, they indirectly achieve the minimization of the sum-of-squares through their operations. While BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [ZRL96] constructs a tree-like data structure to represent data points and merges similar clusters, CURE [GRS98] (Clustering Using Representatives) takes a different approach: it selects a fixed number of well-scattered points to represent each cluster.

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [ZRL96] is a scalable clustering algorithm specifically designed to work with large datasets. It uses a tree structure to represent the data, enabling it to cluster large datasets in a single scan while limiting memory usage.

Here are the basic steps of the BIRCH Algorithm 14:

1. Data Representation. BIRCH represents data using a tree structure known as the Clustering Feature (CF) tree. Each node in the tree will contain one or more Clustering Features (CFs), which are summaries of data points. In non-leaf nodes, each CF summarizes a different child node (and its subtree), helping to guide the search through the tree. In leaf nodes, each CF represents a different subcluster of the data, and the leaf nodes are where the actual clustering takes place. This hierarchical structure is

what allows BIRCH to handle large datasets efficiently. A CF is a triplet (N, LS, SS), where N is the number of data points, LS is the linear sum of the data points, and SS is the square sum of the data points. Each node also has a threshold, T, which sets the maximum diameter or radius of the subclusters it can hold. The branching factor B is a parameter that specifies the maximum number of children or CFs that a node in the CF tree can have. The CF Tree maintains a balanced tree structure, much like a B-tree;

2. Building the CF Tree. When a new point is added, the process starts from the root of the CF tree. The tree is descended by continually choosing the child CF (subtree) that is closest to the new point, until a target leaf node is reached. This node houses several CFs, each of which signifies a cluster of points. The algorithm then looks for the CF in the leaf node that is closest to the new point. If this CF is within the radius threshold T from the new point, the point is absorbed into the CF by updating its summary statistics. Conversely, if the closest CF is beyond the threshold T from the point, a new CF is created for the point. However, if adding this new CF causes the leaf's capacity to exceed its branching factor B, the leaf node is split into two. The CFs are distributed between the two new nodes in a way that minimizes the intra-cluster distance and maximizes the inter-cluster distance. If a leaf splits, the parent node must accommodate a new CF representing the new leaf. If this causes the parent node to exceed its CF limit, defined by the branching factor B, it also has to be split. This potential splitting can propagate up to the root node, if necessary. This is similar to insertions in a B-tree;

3. Clustering. Once the CF Tree is built, it can be used to generate clusters. There are several ways to do this. One common approach is to treat each leaf entry as a cluster, resulting in a large number of small clusters. Then, a second pass can be made using a traditional clustering algorithm (like agglomerative hierarchical clustering) to combine these small clusters into larger ones.

Algorithm 14: BIRCH Clustering Algorithm

Function `BIRCH(`X, T, B`)`**:**
  Input : Data points $X = \{x_1, \ldots, x_m\}$;
    Threshold T;
    Branching factor B.
  Output: Clustering Feature Tree T;
    Point-to-cluster assignments $Y = \{y_1, \ldots, y_m\}$.
  Initialize an empty CF Tree $\mathcal{T}$;
  foreach x_i *in* X **do**
    Let $\mathcal{N}$ be a leaf node, which is the result of recursively traversing the CF Tree from the root $\mathcal{R}$, at each step selecting the child node $\mathcal{C}$ that minimizes $d(\mathcal{C} + x_i, \mathcal{C})$;
    if $\exists$ *CF* j *in* $\mathcal{N}$ *with* $d(j + x_i, j) \leq T$ **then**
      $j \leftarrow j + x_i$;
    end
    else
      Append new CF $j = x_i$ to $\mathcal{N}$;
      if $|\mathcal{N}| > B$ **then**
        Split $\mathcal{N}$ into two nodes, propagate upwards, splitting nodes as needed;
      end
    end
  end
  for $i = 1$ **to** m **do**
    $y_i \leftarrow$ index of CF j in $\mathcal{T}$ minimizing $d(x_i, j)$;
  end
  return T, Y

In the algorithm, $d(j + x_i, j)$ stands for the increase in the diameter of CF j when x_i is added. This is defined as $d(j + x_i, j) = ||\frac{LS_{j+x_i}}{N_{j+x_i}} - \frac{LS_j}{N_j}||$, where LS and N represent the linear sum of points and the number of points in a CF, respectively. Similarly, $d(x_i, j)$ represents the distance between x_i and the centroid of CF j and is defined as $d(x_i, j) = ||x_i - \frac{LS_j}{N_j}||$. Therefore, during the process of building the CF tree, the CF in each node that results in the smallest increase in diameter when a new point is added is chosen.

In its time, BIRCH revolutionized the representation of data points for clustering by introducing the innovative concept of using a CF tree. BIRCH is considered advantageous for handling large datasets due to its utilization of this memory-efficient data structure.

However, it is not without its limitations. For example, it is sensitive to the order of the data points, as the CF tree can look different depending on the order in which data points are inserted. Additionally, BIRCH requires careful parameter selection for each new dataset: radius threshold T and branching factor B. Finally, BIRCH may struggle with datasets that have a large number of subclusters, as it can become computationally expensive to traverse the clustering tree. Studies have demonstrated that CURE is superior to BIRCH in big data conditions [GRS98]. As a result, we chose not to include BIRCH in our experiments.

CURE (Clustering Using REpresentatives) [GRS98] is a hierarchical clustering algorithm designed with the intention of overcoming the limitations of existing clustering algorithms like K-means and hierarchical clustering, particularly in their ability to handle large datasets and to identify clusters of arbitrary shapes and sizes. It takes a more robust approach to cluster analysis by using multiple representative points instead of a single centroid or medoid to characterize clusters. This way, it captures the shape and variability of a cluster more effectively. The pseudocode of the BIRCH algorithm is presented in Algorithm 15.

The CURE algorithm begins by taking a random sample of size s from the dataset X. This sample is partitioned into $p = s/(f \cdot k \cdot q)$ partitions, each of size $q \cdot k$. Hierarchical clustering is then applied to each partition to form q clusters. Note that each partition is processed independently, making the algorithm amenable to embarrassingly parallel implementation. For each cluster, the c representative points, which are the points furthest from the centroid, are selected and moved towards the geometric center of the cluster by a factor of α. Each cluster, along with its adjusted representative points, is added to a pool P. Hierarchical clustering is then applied to the pool P, where the distance between two clusters is defined as the minimum distance between their representative points. This process continues until the desired number of k clusters is reached. Each data point in the original dataset X is then assigned to the closest cluster centroid. The output is the set of k clusters and the assignments of data points to these clusters.

The outcome is a set of clusters that can be of different sizes and shapes, making CURE a more flexible and robust option compared to traditional hierarchical clustering methods.

Algorithm 15: CURE Clustering Algorithm

Function `CURE`(X, k, c, s, α, q, f)**:**
  Input : Data points $X = \{x_1, \ldots, x_m\}$;
    Desired number of clusters k;
    Number of representatives c;
    Sample size s;
    Shrink factor α;
    Partition constant q;
    Parameter for balancing f;
  Output: Cluster centroids $C = \{c_1, \ldots, c_k\}$;
    Point-to-cluster assignments $Y = \{y_1, \ldots, y_m\}$.
  $s_X \leftarrow$ a random sample of size s from X;
  $p \leftarrow s/(f \cdot k \cdot q)$;
  Partition s_X into p partitions each of size $q \cdot k$;
  Initialize an empty pool P;
  foreach *partition in* s_X **do**
    Let $\mathcal{Q}$ be q clusters obtained from hierarchical clustering on the partition;
    foreach *cluster* Q *in* $\mathcal{Q}$ **do**
      Let R be the c furthest points from the centroid of Q;
      $R' \leftarrow R + \alpha \cdot (cent(Q) - R)$;
      Add $\{Q, R'\}$ to P;
    end
  end
  Perform hierarchical clustering on P using minimum distance between representative points until k clusters are formed, denoted by $\mathcal{K}$;
  foreach *cluster* K *in* $\mathcal{K}$ **do**
    Set c_K as the centroid of K;
  end
  foreach *point* x_i *in* X **do**
    $y_i \leftarrow$ cluster K in $\mathcal{K}$ that minimizes $d(x_i, c_K)$;
  end
  return $\mathcal{K}$, Y

However, as it involves computation of distances between pairs of representative points, the complexity can be high, making it computationally intensive for large datasets.

The outcome is a set of clusters that can be of different sizes and shapes, making CURE a more flexible and robust option compared to traditional hierarchical clustering methods. However, as it involves computation of distances between pairs of representative points, the complexity can be high, making it computationally intensive for large datasets.

5.3.4 Canopy Clustering

The Canopy Clustering algorithm, as outlined in [MNU00], serves as a preliminary step for more complex clustering algorithms. It aims to streamline the dataset size, enhancing the speed of subsequent clustering processes.

The crux of the Canopy Clustering algorithm is the creation of "canopies". The algorithm begins by randomly choosing a point in the dataset, which then becomes the center of a new canopy. From there, the algorithm measures the distance from the center point to every other point in the dataset. Two thresholds are predefined for this process: $T1$ and $T2$, where $T1 > T2$. All points within the $T1$ distance from the center point are considered part of the canopy. But only those points within the $T2$ distance cannot become centers of other canopies. Therefore, a point can belong to multiple canopies, but canopies have unique center points. This process of canopy creation continues until all points in the dataset have either been used as the center of a canopy or are within $T2$ distance of an existing canopy center. The pseudocode of the canopy clustering algorithm is provided in Algorithm 16.

The result is a set of overlapping canopies that give a rough initial grouping of the data points. More computationally intensive clustering, such as K-means, can then be applied to the points within each canopy separately, significantly reducing the computation time. This is particularly valuable when the dataset is large, as it allows for more sophisticated clustering algorithms to be used without an excessive computational cost. Also, the canopy method can be particularly useful when dealing with text data, as its low computational requirements and allowance for overlap make it well-suited to handling the high dimensionality and ambiguity

that are characteristic of text data.

Despite these advantages, Canopy Clustering is not without its drawbacks. The quality of the canopies, and hence of the final clustering result, is highly dependent on the chosen distance measure and the values of the $T1$ and $T2$ thresholds. The latter, in particular, can be difficult to set correctly, as they require a good understanding of the data's scale and distribution. Furthermore, since the canopy clustering only provides an initial partitioning of the data, it needs to be used in conjunction with another clustering algorithm to obtain a final clustering result. This means that the overall clustering process may still be quite computationally intensive, especially if the subsequent clustering algorithm is a complex one. Finally, the algorithm does not guarantee optimal clustering, and the final clustering result can be affected by the order in which the data points are processed.

Xia et al. proposed a parallel adaptive Canopy-K-means clustering algorithm for big data, leveraging the MapReduce cloud computing model [XNH20]. While it promises to manage the complexities of large datasets with a unique adaptive process for selecting the similarity threshold, there is ambiguity in its description. Despite their claims of a MapReduce implementation, our attempts to reproduce their algorithm were thwarted by this unclear adaptive procedure, which was the key step in their algorithm. Consequently, we unable to successfully replicate or verify the assertions made in the paper.

Beyond hardware acceleration and memory efficiency, K-means optimization can be achieved through analytical refinement of the underlying mathematical operations, as explored in the next subsection.

5.3.5 Triangle inequality

During K-means clustering, it can be observed that not all points alter their cluster membership. By using some logic to identify and label these points as fixed, the algorithm can avoid the need to recalculate cluster distances for them. This optimization can significantly reduce computation time. The triangle inequality is widely employed to identify points that remain fixed to their respective centroids across iterations [Elk03].

Algorithm 16: Canopy Clustering

```
1  Function Canopy_clustering(X, T1, T2):
       Input  : Feature vectors X ∈ ℝ^{m×n};
                Distance threshold for canopy inclusion T1;
                Distance threshold for point removal T2;
       Output: Set of canopies C.
2      Initialize an empty set of canopies C;
3      while X is not empty do
4          Randomly select a point x from X;
5          Create a new canopy c with center x;
6          for each point y in X do
7              if distance(x, y) < T1 then
8                  Add y to c;
9              end
10             if distance(x, y) < T2 then
11                 Remove y from X;
12             end
13         end
14         Add c to C;
15     end
16     return C
```

Moodi et al. introduced an optimized version of the K-means algorithm that utilizes the triangle inequality to reduce processing time [MS22]. Their approach applies the A-means algorithm according to Early Classification (EC), which stabilizes points to their respective clusters based on the probability of membership change in subsequent iterations. This reduces the computational time required for assigning points to clusters, especially for large datasets. However, significant movement of centroids in some iteration can lead to inaccurate results. To address this issue, the authors proposed a method to reintroduce stabilized points into computation if their distances from the centroids exceed the cluster radius. This ensures that the approach achieves significant computational savings without compromising accuracy. We have included this algorithm into our experiments under the name IK-means. The pseudocode of the IK-means algorithm is shown in Algorithm 17.

The IK-means algorithm reduces the computation of the K-means algorithm by reducing the number of points that need to be reassigned to clusters in each iteration. This is achieved by introducing the concept of exclusion, where points that are very likely to remain in their current cluster are excluded from further consideration in the algorithm. The exclusion is based on the triangle inequality, which can help estimate whether a point is likely to change its cluster assignment or not, thus reducing the number of distance computations in the algorithm.

The triangle inequality can also be leveraged to accelerate the K-means local search within the Big-means algorithm, resulting in a substantial increase in efficiency. This demonstrates that Big-means is a highly versatile algorithm within the realm of MSSC algorithms, capable of applying optimization at almost every step and addressing nearly all challenges of big data clustering, showcasing its all-round excellence.

The following subsection explores another avenue for optimizing K-means: improving initialization efficiency through strategic sampling.

Algorithm 17: IK-means Clustering

Function `IK_means`(X, k, T, E)**:**
- **Input :** Feature vectors $X \in \mathbb{R}^{m \times n}$;
  Number of clusters k;
  Number of iterations T;
  Exclusion evaluation interval E;
- **Output:** Centroids $C \in \mathbb{R}^{k \times n}$.

Initialize all k centroids C randomly from X;
Initialize the assignment vector $A \in \mathbb{R}^m$ with zeros;
Initialize the cluster radii $R \in \mathbb{R}^k$ with zeros;
Initialize the centroid shifts $S \in \mathbb{R}^k$ with zeros;
Initialize the exclusion vector $E \in \mathbb{R}^m$ with zeros;
for $t = 1$ **to** T **do**
  if $t \mod E = 0$ **then**
    for $i = 1$ **to** m **do**
      if $E[i] = 1$ ***and*** $\|C[A[i]] - X[i]\| > R[A[i]]$ **then**
        $E[i] \leftarrow 0$;
      end
    end
  end
  for $i = 1$ **to** m **do**
    if $E[i] = 0$ **then**
      Compute distances d to all centroids C;
      Sort d and get the indices of the two nearest centroids $idx1$, $idx2$;
      $A[i] \leftarrow idx1$;
      if $t \geq 2$ ***and*** $\|d[idx1]^2 - d[idx2]^2\| > S[idx1] + S[idx2]$ **then**
        $E[i] \leftarrow 1$;
      end
    end
  end
  if $\sum E = m$ **then**
    break;
  end
  for $j = 1$ **to** k **do**
    Compute new centroid $C_{\text{new}}[j]$ as the mean of X where $A = j$;
    Compute new radius $R[j]$ as the maximum distance from $C_{\text{new}}[j]$ to X where $A = j$;
    Compute centroid shift $S[j]$ as $\|C[j] - C_{\text{new}}[j]\|$;
    $C[j] \leftarrow C_{\text{new}}[j]$;
  end
end
return C

5.3.6 Sampling-based initialization

Initializing the K-means algorithm on a small random subset of the data and then applying it to the entire dataset is a powerful strategy for optimizing K-means in the big data context. This approach addresses scalability challenges, enabling the use of more advanced initialization methods for improved clustering performance.

Density-based algorithms, like DBSCAN or OPTICS, form clusters based on areas of high data point density, separated by areas of low density. They can discover clusters of arbitrary shapes and sizes, and are capable of handling noise and outliers. They are generally less sensitive to the initialization issue, unlike K-means, which can converge to different solutions depending on the initial centroids. Density-based methods can be slower and more computationally intensive, especially on large datasets. They also often require tuning parameters related to the density concept, which can be difficult in practice. However, applying a density-based algorithm, such as DBSCAN or OPTICS, to a smaller random sample from the dataset may be computationally justified.

Since a random sample is usually relatively small, more computationally complex algorithms can be applied to it. For instance, paper [Die+17] proposed a new clustering algorithm for big data that leverages the idea of applying a density-based algorithm, DBSCAN, to a small random sample to initialize the global clustering process. The resulting centroids from the sample are used to initialize the K-means algorithm on the entire dataset, which can lead to a faster convergence and higher quality clusters than initializing K-means randomly. A representative algorithm employing this approach is the K-means Clustering Data Science and Engineering Solution, abbreviated as CluDataSE. Its pseudocode is shown in Algorithm 18. CluDataSE provides a way to balance accuracy and efficiency in clustering large datasets.

The multi-start K-means algorithm [FS19] is a variant of the K-means algorithm that aims to improve the quality of the produced clusters by running the K-means algorithm multiple times with different initializations. The algorithm randomly initializes the cluster

Algorithm 18: CluDataSE Clustering

Function `CluDataSE`(X, k, s, eps, min_pts)**:**
  Input : Dataset $X = \{x_1, \ldots, x_m\}$;
    Number of clusters k;
    Sample size s;
    Distance threshold eps;
    Minimum points min_pts;
  Output: Centroids C;
    Cluster assignments Y for X.
  $S \leftarrow$ random sample of size s from X;
  $C \leftarrow \varnothing$;
  while $|C| < k$ **do**
    $C \leftarrow C \cup \text{DBSCAN}(S,\ eps,\ min\_pts)$;
  end
  $C \leftarrow$ K-means++$(C,\ k)$;
  $C \leftarrow$ K-means$(X,\ C)$;
  $Y \leftarrow$ Assign each point in X to the nearest centroid in C;
  return $C,\quad Y$

centers multiple times and runs the K-means algorithm on each initialization. The final set of clusters is selected from among the various runs based on the lowest sum of squared distances between the data points and their respective cluster centers.

The Forgy K-means algorithm [Pen99] is a simple and efficient variant of the K-means algorithm. Unlike random initialization, where centroids are chosen as random locations in the space, Forgy initialization randomly selects k data points from the dataset to serve as the initial centroids. This approach ensures that the initial cluster centers are actual data points, making them more representative of the dataset and potentially leading to faster convergence of the algorithm. This selection strategy leads to the initial cluster centers that are more likely to be close to the true cluster centers since randomly picked data points tend to be close to regions with high density. Meanwhile, using random space locations can sometimes lead to initial centroids that are not representative of the dataset or that are too far away from the true clusters.

The Big-means algorithm exemplifies the use of K-means++ initialization on a random

subsample of the dataset. The algorithm then employs an iterative process to improve the current best set of centroids using local search on various subsequent subsamples, thereby reducing the dependence of final results on the accuracy of initialization. As a result, different initialization schemes can also be used on the first sample in Big-means.

The following subsection reveals that sampling has far-reaching benefits, extending beyond initialization to enhance K-means iterations themselves.

5.3.7 Approximation techniques

Approximation techniques aim to create a compressed or simplified representation of the original dataset while preserving its essential structure and properties. These techniques allow clustering algorithms to efficiently process large datasets while striving to minimize any loss in accuracy.

Lightweight coreset [BLK18] is a technique used to generate a smaller, weighted subset of the original dataset that preserves its essential properties. Coresets can be utilized to improve the computational and memory efficiency of big data clustering algorithms by reducing the number of data points processed. They are useful in big data clustering as they enhance scalability and can be adapted to different clustering algorithms. Despite these advantages, coresets have some limitations, such as a slight loss of accuracy due to approximation, complexity in constructing them, sensitivity to the data distribution, and the need for domain expertise in parameter tuning for optimal performance. Overall, coresets provide a valuable approach for handling large-scale clustering tasks while addressing resource constraints.

Essentially, a coreset [Moh+18] is small summary of the dataset with the property that the solution of a clustering problem on this summary is provably competitive with the solution on the full dataset. More specifically, given $\varepsilon > 0$, the weighted set C is called an (ε, k)-lightweight coreset of X if for any subset $Q \subset \mathbb{R}^n$ of cardinality at most k the following holds:

$$|f(X,Q) - f(C,Q)| \leq \frac{\varepsilon}{2} f(X,Q) + \frac{\varepsilon}{2} f(X, \{\mu(X)\})$$

where $\mu(X)$ stands for the mean of dataset X.

Lightweight coresets can be constructed as follows. Paper [Moh+18] provides a simple joint probability distribution over the elements in X, which is then used for construction of a lightweight coreset. For each $x \in X$ the probability is defined as

$$q(x) = \frac{1}{2}\frac{1}{|X|} + \frac{1}{2}\frac{\|x-\mu\|^2}{\sum_{x' \in X}\|x'-\mu\|^2} \tag{5.2}$$

A lightweight coreset is constructed by sequentially sampling new centroids from the dataset points X according to the probability distribution (5.2). Computing the above joint probability distribution takes two full passes through the whole dataset X. This fact might make the coreset technique less efficient for true big data. Moreover, sampling from the distribution (5.2) without replacement has the time complexity of $\mathcal{O}(m\log(s))$, where s is the total number of points to be sampled. Thus, although the algorithm for construction of lightweight coresets is quite simple from the standpoint of "less is more", it still does not satisfy our desired requirements for "true big data" algorithms.

According to the experimental results in [BLK18], lightweight coresets did not outperform the regular K-means++ on the full dataset with respect to the objective function (1.1). Lightweight coresets only showed a greater reduction in the objective function value (1.1) compared to the "naive" uniform sampling and other compression methods. Using the regular K-Means++ on the full dataset still provided the best results in comparison to the naive sampling and the coreset technique.

As a promising technique for big data applications, a clustering algorithm based on applying the K-means++ seeding to a lightweight coreset was included into experiments under the name LW-coreset. The pseudocode for this method is provided in Algorithm 19.

Another efficient yet effective approximation method, employed by the Big-means algorithm, involves taking a simple uniform sample from the available data X. This approach has a constant time complexity of $\mathcal{O}(1)$ and does not require the calculation of joint probability distributions. Additionally, under a mild assumption on the dataset structure, simple uniform sampling is theoretically guaranteed to produce high-quality solutions, as demonstrated

Algorithm 19: LW-Coreset: Clustering using Lightweight Coreset

Function `LW_coreset_clustering`(X, k, s)**:**
  Input : Dataset X;
    Number of clusters k;
    Coreset size s;
  Output: Cluster centroids $C_k = \{c_1, \ldots, c_k\}$;
    Point-to-cluster assignments Y.
  Compute the mean μ of the dataset X;
  Compute the sum of squared distances $D = \sum_{x' \in X} \|x' - \mu\|^2$;
  for *each* $x \in X$ **do**
    Compute the probability $q(x) = \frac{1}{2|X|} + \frac{1}{2}\frac{\|x-\mu\|^2}{D}$;
  end
  Set C to an empty set;
  for i *in range* s **do**
    Sample a point x from X according to the probabilities $q(x)$, without replacement;
    Add x to C;
  end
  Initialize k centroids C_k from C using K-means++;
  while *centroids* C_k *change* **do**
    Assign each point in C to the closest centroid, forming k clusters;
    Update each centroid to be the mean of the points assigned to its cluster;
  end
  $Y \leftarrow$ Assignment of each point $x_i \in X$ to the closest cluster centroid $c_j \in C_k$;
  return $C_k, \quad Y$

in [HLD23]. This method is optimal according to the "less is more" principle, making it the fastest possible sampling approach computationally.

The next subsection explores the final frontier of K-means optimization, harnessing its power within advanced global optimization metaheuristics or synergizing it with other clustering approaches and meta-ideas.

5.3.8 Hybrid approaches

Hybrid clustering algorithms often combine multiple techniques or algorithms to improve clustering performance, such as using genetic operations to improve initial centroid selection or combining K-means with hierarchical clustering. The choice of the combination of approaches depends on the specific characteristics of the data and the computational resources available. While these approaches can improve clustering quality, they often incur high computational costs due to their increased complexity.

Karmitsa et al. proposed LMBM-Clust in [KBT18], an incremental clustering method designed for very large datasets with a large number of attributes and data points. The method combines the limited memory bundle method (LMBM) [HMM07] as the local search with a method for generating starting centers proposed by Ordin et al. in [OB15]. Experimental results show that LMBM-Clust produces good results in terms of both accuracy and time, only losing to K-means++ in terms of time. The method also requires fewer distance function evaluations than other incremental approaches. However, one major drawback of LMBM-Clust is its high complexity and a large number of hyperparameters. The original implementation of LMBM-Clust provided by the authors was unable to cluster a significant portion of large real-world datasets in our experimental evaluation due to the out-of-memory error [Mus+23].

Gribel et al proposed a hybrid clustering algorithm called HG-means in [GV19], which combines the multi-start K-means approach with genetic operations such as mutation, crossover, and selection to select new initial centroids. The algorithm was shown to produce high-quality clusters and to be faster than many existing algorithms, with the exception of LMBM-Clust.

However, HG-means requires clustering the full dataset in every iteration, which makes the algorithm unsuitable for big data due to its high computational cost.

Mansueto et al. [MS21] proposed a Memetic Differential Evolution Clustering (MDEClust) algorithm for the MSSC problem. The algorithm combines the global optimization framework of differential evolution with the local search procedure of K-means clustering. The MDEClust algorithm evolves a population of solutions through crossover, mutation, and local refinement via K-means. The algorithm ensures diversity and avoids premature convergence by employing an innovative crossover operation based on differential evolution and a roulette wheel selection in the mutation process. The MDEClust algorithm has been shown to be efficient in generating high-quality solutions, especially for datasets with a large number of clusters and high dimensionality. However, the algorithm's performance can be sensitive to the choice of initial population and parameter settings, including population diversity tolerance, maximum number of iterations, crossover parameter, and mutation parameter. While MDEClust outperformed HG-means in terms of clustering accuracy, scalability remains a challenge as the computational effort increases significantly, especially when the mutation operator is used. Additionally, neither MDEClust nor HG-means claim to handle clustering datasets with a large number of input objects m.

In the context of big data, where datasets can be very large and high-dimensional, hybrid clustering algorithms may not be suitable due to their high computational complexity. These algorithms may require extensive memory usage or excessive processing time, making them impractical for large datasets. Furthermore, hybrid algorithms often involve a large number of hyperparameters that need to be carefully tuned, which can be difficult and time-consuming.

In summary, while hybrid clustering algorithms can improve clustering performance, their high computational complexity and extensive hyperparameters make them difficult to apply to large datasets. As a result, simpler and more scalable clustering algorithms, such as K-means and its variants, are often preferred for big data clustering. In particular, we did not consider any of the advanced hybrid MSSC clustering algorithms in our experimental study.

5.4 Other review papers

Recent review papers on scalable and parallel big data clustering methods offer valuable insights into the field's current state. Notably, Mehdi et al.'s review paper [MHE21b] surveyed 101 algorithms and found that most have complex implementations. Moreover, the authors highlight a shift towards deploying algorithms on cloud-based infrastructure, rather than developing new practical methods. This underscores the importance of efficient and scalable clustering approaches in big data analytics.

The survey in [SAA20] concluded that few existing methods support the variety and velocity characteristics of big data. In other words, most of the available parallel clustering algorithms do not handle real-time data, focusing on a single, mostly numerical, type of data. This limits their capability to process big data [DLS20].

Based on the detailed discussion of different parallel algorithms, Mohebi et al. [Moh+16] concluded that the field of parallel big data clustering is still young and open for new research. They argued that parallel data processing can help improve the clustering time of large datasets, but it may degrade the quality and performance. Therefore, the main concern is to achieve a reasonable trade-off between quality and speed in the context of big data.

5.5 Overview

In summary, there are various ways to address the challenges of applying K-means-like techniques to big data. Sampling approaches, like Minibatch K-means, use a random subset of the data for the computation of centroids, thus making the clustering process more efficient. Algorithms like BDCSM use partitioning, where the dataset is divided into chunks, each chunk is clustered separately, and the results are aggregated. Other approaches leverage parallelization and distributed computing to distribute the workload across multiple processors or machines. Some methods, such as IK-means, exploit the properties of geometric spaces, like the triangle inequality, to reduce the computation time. The Big-means algorithm combines both the sampling and parallelization approaches in a unique way that

maximizes not only efficiency, but the accuracy as well.

However, each of the considered methods has its own trade-offs and potential challenges, so it is important to choose the right approach based on the specific requirements and constraints of the given clustering task, characteristics of the data, and the computational resources available. Evaluating different approaches and selecting the one that best fits the problem at hand is crucial for achieving accurate and efficient clustering results. In Section 5.7, we provide an empirical comparison of these methods to understand their performance and efficacy in clustering big datasets.

In essence, only a few algorithms are truly applicable to handling big data: BDCSM and clustering on coresets. In our experiments, the solutions derived from BDCSM consistently underperformed compared to other approaches we evaluated. Alternatively, while the coreset construction procedure does not demand access to the entire dataset, it does necessitate at least two passes through the full dataset. This can be impractical in the context of big data.

The other considered approaches either require multiple passes through the dataset, rely on complex ideas and metaheuristics, exhibit high computational complexities, or suffer from the out-of-memory problem. Furthermore, these approaches struggle to ensure global search during clustering without resorting to additional metaheuristics. These findings underscore the necessity for a novel algorithm capable of addressing these challenges, with the added benefit of adaptable scalability. Scalability is one of the criteria for "true big data" algorithms. These criteria were precisely formulated in Section 3.3.

Our literature review reveals that, to date, there are no simple algorithms that can address the MSSC problem efficiently in big data scenarios while consistently surpassing traditional methods across datasets of varying sizes, from small to large and big. Many authors acknowledge that solving the NP-hard MSSC problem in the context of big data remains an unsolved challenge or a burden for modern computer technologies. Therefore, there is an urgent demand for a novel MSSC algorithm that views big data as an advantage rather than a curse. For widespread practical applicability, the new algorithm should also be simple in terms of its components, computational resources, and hyperparameters.

Also, our findings from the current literature indicate that advanced alternative algorithms often struggle when dealing with large datasets. In most instances, these algorithms are tested on datasets containing a few hundred thousand items at most, and performing clustering tasks on such datasets can take several hours. Even when the literature proposes alternative methods that can handle big data to some extent, like coresets or BDCSM, they typically only approach the performance of standard methods in terms of clustering quality but do not outperform them. In fact, in some cases, the clustering quality can be up to 20% worse compared to standard methods [Moh+18]. For both standard and alternative algorithms, working with big data is often seen as a challenge that needs to be addressed rather than an opportunity to improve clustering results. Therefore, it is of utmost importance to prioritize the development of simpler clustering algorithms that are not only effective for processing big data but can also leverage the advantages of big data to enhance the quality of clustering results.

The LIMA number is a measure of algorithm complexity based on the number of distinct operations in the algorithm [Bri+23]. In this work, we use the LIMA number as a measure of complexity for several clustering algorithms, as shown in Table 5.1. Notably, despite having lowest LIMA numbers, Big-means and BDCSM algorithms yield quite accurate clustering results for big data, suggesting their effectiveness. On the other hand, the IK-means algorithm, despite having the highest LIMA number, exploits the properties of geometric space (triangle inequality) to efficiently reduce the computation time.

While scalable clustering algorithms provide important tools for handling big data, they also highlight the need for further research and development in this area. Future research should address the ongoing challenges of big data, which include dealing with high-dimensional and multi-view data, handling data sparsity and imbalanced data, and improving the quality of clusters in the presence of noise and outliers. It should also consider simplifying the parameter tuning process and enhancing the reproducibility of clustering results.

Algorithm	LIMA number	Input parameters	Algorithmic ingredients
BDCSM	5	1 (p)	Partitioning, clustering, centroid pooling, final clustering, final assignment
Big-means	6	2 (s, T)	Random sampling, conditional reinitialization, centroid update, condition checking and updating, iteration, final assignment
Minibatch K-means	6	2 (T, m)	Initialization, random sampling, assignment, centroid update, iteration, final assignment
K-means++	6	1 (T)	Random selection, distance calculation, probabilistic selection, iteration, cluster assignment, and centroid update
CURE	7	6 (s, f, k, q, c, α)	Random sampling, partitioning, hierarchical clustering, representative selection, geometric transformation, iteration, and cluster assignment
CluDataSE	7	3 (s, eps, min_pts)	Random sampling, density-based clustering, cluster center reduction, k-means clustering, iteration, parameter adjustment, and cluster assignment
LW-Coreset	7	1 (s)	Mean calculation, distance computation, probability computation, sampling, centroid initialization, centroid update, and assignment
IK-means	8	0	Initialization, distance computation, exclusion check, assignment, centroid reinitialization, centroid update, radius update, convergence check

Table 5.1: LIMA numbers of the considered algorithms

5.6 Generalizations, reflections, and practical advices

As the scale and complexity of data increase, it becomes increasingly complex to choose the right big data clustering algorithm, as the traditional choices of clustering algorithms can struggle to work effectively and efficiently at this scale. Therefore, one of the aims of this review paper is to guide and support practitioners in understanding, selecting, and implementing suitable big data clustering methods for their datasets.

5.6.1 The imperative of scalability

A recurring theme in the discussed techniques is the imperative for scalability, as big data presents inherent challenges in volume, velocity, and variety. The sheer size and speed of data generation demand algorithms that are both efficient and scalable, making the use of algorithms like Big-means a necessity.

5.6.2 Complexity and its trade-offs

Big data is not just about size; it is about complexity too. High dimensionality, varying data distributions, presence of noise and outliers, all these factors contribute to the complexity. While techniques like the density-based clustering (DBSCAN), CURE, and the EM algorithm [DLR77] can handle complex distributions, there is always a trade-off to consider between computational complexity, speed, and accuracy. Often, MSSC remains as the only feasible paradigm to use in big data conditions.

5.6.3 Trade-offs in big data clustering

It is essential to understand that there is no all-encompassing algorithm for big data clustering under the MSSC paradigm. The right algorithm depends on the specific characteristics and requirements of the task at hand. For instance, Big-means is well-suited for large and big datasets when both speed and accuracy are essential. On the other hand, hierarchical (Ward's method [Jr63]) or evolutionary (MDEClust) algorithms might be better when accuracy is prioritized and the dataset is small to moderate in size.

The choice of clustering algorithm often involves trade-offs between computational efficiency, accuracy, ability to handle complex distributions, and resilience to noise or outliers. For example, Big-means, Minibatch K-means, and BDCSM are fast, but may not handle complex data distributions. Similarly, while Ward's method and MDEClust may offer higher quality clusters, they can be computationally intensive and more challenging to tune.

5.6.4 Understanding algorithmic strengths and weaknesses

Big-means offers both effectiveness and efficiency in big data clustering, addressing most MSSC challenges. However, it assumes convex and isotropic clusters, which may not always hold in real-world data. Moreover, while it does not explicitly handle noise and outliers, random sampling in the algorithm partially mitigates this limitation.

Hierarchical clustering, specifically Ward's method, provides an interpretable dendrogram and does not require the number of clusters to be predefined. However, it can be extremely computationally expensive for big data.

Density-based clustering, such as DBSCAN, can discover clusters of arbitrary shape and is resilient to outliers. It can be a powerful tool for datasets with complex distributions but tuning its parameters (eps and $MinPts$) can be challenging. Also, its computational complexity can be as high as quadratic with respect to the overall number of data points.

Evolutionary clustering algorithms like MDEClust have the potential to reveal insightful clusters through multiple generations of solutions, but they are quite computationally demanding and may not be practical for large-scale data.

HG-means, as a hybrid algorithm, tries to balance the strengths of hierarchical and K-means clustering but inherits some of their weaknesses too, including the sensitivity to initial centroids.

5.6.5 Guiding the choice of a big data clustering algorithm

We introduce a flowchart (Figure 5.4), a practical tool for selecting an initial clustering algorithm based on data size, number of clusters, distribution, and speed-accuracy trade-off, serving as a valuable aid in big data analysis. However, it is important to note that these are guidelines and not definitive rules. Depending on the specific circumstances, it may be necessary to experiment with multiple algorithms or a combination of them. The final decision may require experimentation and fine-tuning, based on the specifics of a task.

The decision should also be guided by the data characteristics and the specific require-

ments of the task. For large and big datasets, Big-means offers a rapid clustering solution with high accuracy, making it a suitable choice when speed and accuracy are equally important. For a small to moderate-sized dataset where accuracy and quality of clusters are paramount, hierarchical clustering (Ward's method), DBSCAN, or MDEClust may be preferable. For complex distributions, DBSCAN, CURE, or the EM algorithm would be ideal, while simpler distributions could be efficiently handled by Big-means or K-means++.

Our hope is that the presented flowchart-based approach to the selection of clustering algorithms can be a practical tool to guide data scientists in the selection of the most suitable algorithm for their specific application and dataset.

5.6.6 The role of experimental validation (experimentation and iteration)

It is crucial to validate the choice of a clustering algorithm through experimentation. Apply the chosen algorithm to a subset of your data, analyze its performance and appropriateness for your task, and adjust accordingly. This iterative process allows to refine the choice and find the most suitable algorithm for the specific task.

5.6.7 Summary

In summary, the complexity of big data clustering requires a flexible, informed approach. By understanding the strengths, weaknesses, and trade-offs of different algorithms and techniques, and by adopting an experimental, iterative mindset, practitioners can effectively navigate the challenges of big data clustering.

5.7 Experimental evaluation

For the information on the used datasets, software and hardware, experimental methodology, evaluation metrics, as well as other implementation details, we refer the reader to Section 4.6 of Chapter 4. In this chapter, the design of the experiments is identical to Chapter 4, except for some minor differences that we will describe below.

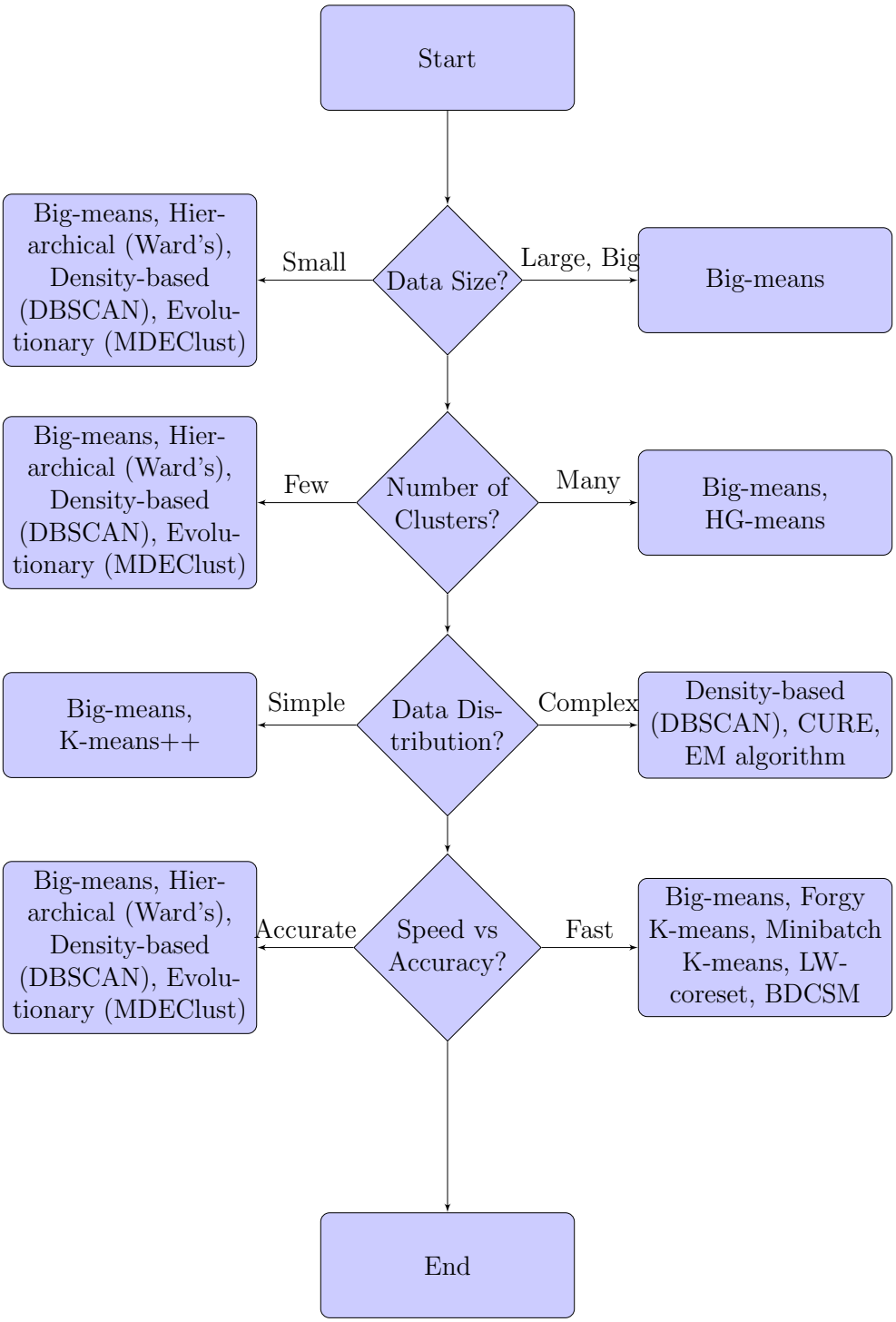


Figure 5.4: Flowchart of big data clustering algorithm selection

5.7.1 Range of tested algorithms

We decided to test the algorithms presented in the LIMA Table 5.1: BDCSM, Big-means, Minibatch K-means, K-means++, CURE, CluDataSE, LW-Coreset, IK-means.

5.7.2 Time metric

In the experiments of this chapter, for each choice of a dataset X and a number of clusters k, the resulting CPU time t is measured differently from Chapter 4.

For Big-means, CPU time t is considered to be the time of the last change of the incumbent solution C_w for the worker w that achieved the best value of the objective function $f(C_w, S_w)$ on some sample S_w. For all other algorithms, CPU time t is the total time required to cluster the given dataset X into k clusters, excluding the time of computing point-to-cluster assignments. All values of t should be treated as having the unit of a second.

5.7.3 Other implementation details

We used the hybrid-parallel version of Big-means (HPClust-hybrid) in this chapter's experiments since this version was found to result with the best performance for Big-means in Chapter 4.

5.7.4 Experimental results

The total number of conducted experiments reached 29,464.

Tables 5.2 – 5.3 exhibit the experimental results. Appendix C provides detailed information on all experiments conducted. In the tables of Appendix C showing clustering details, n_d denotes the total number of distance function evaluations performed by an algorithm across different executions and all choices of k, while n_s stands for the total number of processed samples by the Big-means algorithm.

Table 5.2: Relative clustering accuracies (ϵ, in percentage) resulting from the comparison of the selected clustering algorithms

Dataset	Big-means				IK-means				BDCSM				Minibatch K-means			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	2/7	0.02	**0.08**	**0.18**	0/7	0.94	1.95	5.41	0/7	0.6	1.84	4.81	0/7	0.41	0.81	1.81
HEPMASS	2/7	**0.02**	**0.12**	**0.18**	0/7	1.58	2.24	4.18	0/7	0.89	1.61	3.51	0/7	0.53	1.06	2.22
US Census Data 1990	4/7	**0.34**	**1.24**	**3.19**	1/7	4.63	84.2	308.78	0/7	15.32	90.94	279.53	1/7	0.9	2.31	6.96
Gisette	0/7	-0.46	-0.39	-0.3	0/7	-0.04	0.34	1.03	0/7	-0.47	-0.42	-0.32	0/7	-0.25	-0.08	0.39
Music Analysis	2/7	0.4	0.83	**1.87**	0/7	4.84	11.55	22.61	0/7	1.5	4.47	31.86	0/7	0.58	2.43	12.0
Protein Homology	1/7	0.49	0.97	**1.75**	0/7	73.47	169.83	205.07	0/7	6.38	21.85	46.47	0/7	56.02	101.04	176.63
MiniBooNE Particle Identification	3/7	**-0.08**	**0.01**	823309.91	0/7	40846.28	111022.79	311782.51	0/7	2.62	2.84	110987.1	0/7	2810404.23	2826338.32	2831962.18
MiniBooNE Particle Identification (normalized)	0/7	0.29	0.59	**1.44**	0/7	73.84	182.29	246.15	0/7	2.59	8.99	1252.0	0/7	136.31	467.9	537.05
MFCCs for Speech Emotion Recognition	2/7	0.1	**0.36**	**1.03**	0/7	3.87	11.07	25.01	0/7	1.79	19.9	36.15	0/7	0.6	2.35	9.78
ISOLET	6/7	**0.0**	**0.3**	**0.65**	0/7	1.24	2.68	7.54	0/7	0.37	1.02	2.59	0/7	0.4	1.75	3.2
Sensorless Drive Diagnosis	6/7	**-0.43**	**-0.21**	**6.23**	0/7	242.17	672.88	706.19	0/7	154.89	182.9	224.55	0/7	630.63	674.68	694.06
Sensorless Drive Diagnosis (normalized)	6/7	**0.27**	**1.12**	**4.15**	0/7	2.52	8.26	38.27	0/7	3.19	10.66	52.86	0/7	0.53	4.05	10.37
Online News Popularity	5/7	**0.35**	**1.51**	**5.79**	0/7	30.14	95.91	229.91	0/7	13.2	33.77	121.0	0/7	8.5	16.5	37.86
Gas Sensor Array Drift	5/7	**-0.02**	**0.66**	**3.92**	0/7	22.26	77.74	192.13	0/7	8.58	25.99	45.77	0/7	3.06	11.0	26.79
3D Road Network	0/7	0.04	0.24	0.97	0/7	17.08	109.16	407.92	0/7	6.0	40.57	141.57	0/7	0.64	2.76	10.79
Skin Segmentation	5/7	**-0.1**	**0.93**	**4.23**	0/7	11.16	32.25	77.69	0/7	5.34	20.97	82.67	0/7	0.18	3.86	13.63
KEGG Metabolic Relation Network (Directed)	4/7	**-0.32**	**0.28**	**4.74**	0/7	1225.29	2024.58	2324.8	0/7	94.23	96.41	109.05	0/7	1627.77	1859.2	1980.12
Shuttle Control	7/8	**-0.15**	**2.58**	**47.68**	0/8	369.96	558.74	958.42	0/8	136.81	174.37	239.82	0/8	1116.94	1230.13	1300.53
Shuttle Control (normalized)	4/8	0.61	**1.65**	**4.63**	0/8	2.99	24.07	79.56	0/8	8.72	27.68	111.97	1/8	0.95	4.91	18.38
EEG Eye State	4/8	**0.53**	**0.56**	**42.33**	0/8	93292.5	706326.79	1350191.25	0/8	3.85	937883.36	1020813.95	0/8	2438049.8	2593558.8	2639777.84
EEG Eye State (normalized)	8/8	**-0.06**	**-0.01**	**19.56**	0/8	170.58	572.7	692.01	0/8	159.97	572.33	691.21	0/8	904.72	969.67	983.49
Pla85900	4/7	0.06	**0.18**	**0.67**	0/7	4.61	12.9	39.93	0/7	3.4	11.56	45.87	0/7	1.14	3.42	8.06
D15112	4/7	0.08	**0.19**	**0.62**	0/7	6.55	17.84	47.33	0/7	0.38	1.59	7.84	0/7	1.15	3.48	10.32
Overall Results	84/165	**0.09**	**0.6**	35802.84	1/165	5930.8	35740.12	72547.55	0/165	27.4	40836.31	49362.25	2/165	228388.95	235880.88	238155.85

Dataset	K-means++				CURE				CluDataSE				LW-coreset			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	3/7	**-0.01**	0.13	1.44	0/7	20.59	28.29	35.69	3/7	-0.0	0.14	0.83	0/7	0.04	0.25	0.76
HEPMASS	2/7	0.04	0.21	0.69	0/7	24.12	26.32	29.19	3/7	0.08	0.17	0.73	0/7	0.13	0.55	1.01
US Census Data 1990	1/7	1.4	4.95	84.97	0/7	17.55	24.92	33.33	1/7	4.1	9.59	27.51	0/7	9.11	118.25	245.89
Gisette	2/7	**-0.52**	**-0.48**	**-0.39**	0/7	8.87	9.51	9.85	5/7	**-0.52**	**-0.48**	**-0.39**	0/7	-0.36	-0.27	-0.11
Music Analysis	4/7	**-0.03**	**0.55**	7.52	0/7	26.74	44.31	79.79	2/7	0.0	2.19	2.96	0/7	0.53	1.88	10.36
Protein Homology	6/7	**0.02**	**0.4**	15.15	0/7	216.6	243.48	286.52	2/7	14.85	14.9	15.01	0/7	0.44	14.68	15.54
MiniBooNE Particle Identification	4/7	-0.06	0.48	**24.53**	0/7	2832329.21	2832343.36	2832361.67	2/7	5.71	40994.64	40994.98	0/7	1.11	4.11	28.27
MiniBooNE Particle Identification (normalized)	5/7	**-0.05**	0.73	101.48	0/7	22.75	33.94	155.39	4/7	0.09	**0.56**	2.07	0/7	0.39	2.25	4.4
MFCCs for Speech Emotion Recognition	3/7	**0.0**	0.72	2.34	0/7	22.72	39.4	71.09	4/7	0.75	1.25	3.22	0/7	0.37	1.75	4.5
ISOLET	1/7	0.02	0.63	2.15	0/7	25.42	38.99	48.75	0/7	0.35	1.34	2.31	0/7	0.35	1.45	3.73
Sensorless Drive Diagnosis	1/7	-0.38	10.76	63.17	0/7	739.52	755.39	778.76	0/7	122.72	162.38	224.34	0/7	21.94	128.63	145.29
Sensorless Drive Diagnosis (normalized)	1/7	0.47	4.27	22.13	0/7	14.23	47.3	97.82	0/7	1.25	6.06	26.66	0/7	2.65	11.54	57.73
Online News Popularity	1/7	1.07	6.71	29.74	0/7	81.69	137.86	210.37	2/7	9.29	17.71	30.4	0/7	1.86	16.81	77.2
Gas Sensor Array Drift	2/7	0.32	3.5	22.34	0/7	57.44	82.36	135.38	1/7	8.08	16.14	26.17	0/7	1.63	8.12	28.57
3D Road Network	7/7	**-0.0**	**0.01**	**0.54**	0/7	44.47	100.47	215.66	5/7	0.23	0.23	4.01	0/7	0.78	4.68	12.73
Skin Segmentation	1/7	0.29	5.24	17.35	0/7	27.42	95.62	165.49	2/7	6.68	15.73	23.83	0/7	4.46	15.72	34.25
KEGG Metabolic Relation Network (Directed)	2/7	0.22	4.84	66.24	0/7	2817.93	2944.16	3240.08	1/7	94.27	94.27	95.61	0/7	11.19	64.11	77.88
Shuttle Control	1/8	2.76	16.01	73.7	0/8	1336.47	1339.01	1341.97	0/8	139.1	189.52	232.24	0/8	8.08	45.09	182.03
Shuttle Control (normalized)	1/8	1.03	7.28	38.15	0/8	33.8	88.43	293.11	3/8	**0.39**	3.98	51.82	0/8	7.31	32.76	126.54
EEG Eye State	3/8	0.54	4.3	51.77	0/8	2641365.35	2641430.58	2641645.46	1/8	854977.62	1020813.06	1020813.77	0/8	1.29	124.33	35642.74
EEG Eye State (normalized)	0/8	**-0.06**	23.01	51.12	0/8	980.93	993.82	1009.74	0/8	462.46	572.78	620.43	0/8	59.44	305.97	564.1
Pla85900	2/7	**-0.01**	0.45	2.09	0/7	17.49	40.43	71.38	2/7	0.0	0.34	1.99	0/7	0.2	1.02	3.21
D15112	3/7	**0.01**	0.74	3.21	0/7	13.76	33.03	62.42	2/7	0.5	2.85	4.79	0/7	0.51	2.21	5.43
Overall Results	56/165	0.31	4.15	**29.63**	0/165	238271.52	238300.91	238364.3	45/165	37210.78	46213.89	46226.32	0/165	5.8	39.39	1620.52

Table 5.3: Total clustering times (t, in seconds) resulting from the comparison of the selected clustering algorithms

Dataset	Big-means				IK-means				BDCSM				Minibatch K-means			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	2/7	15.14	26.27	38.34	0/7	31.44	32.08	34.0	1/7	61.42	74.85	96.2	4/7	**9.62**	**15.7**	**20.46**
HEPMASS	0/7	**5.05**	19.84	27.32	0/7	110.12	112.59	117.47	2/7	32.22	35.01	37.59	0/7	10.36	12.49	16.72
US Census Data 1990	2/7	**0.39**	2.3	**3.04**	0/7	26.46	28.34	30.31	2/7	4.07	4.45	5.3	3/7	0.75	**1.78**	3.38
Gisette	0/7	15.75	19.6	26.08	4/7	6.44	6.8	**7.42**	0/7	21.4	34.54	68.3	3/7	**3.43**	**6.57**	11.64
Music Analysis	0/7	1.53	5.29	8.66	0/7	4.39	4.68	5.0	1/7	5.4	6.9	9.94	5/7	**0.64**	**1.34**	**2.75**
Protein Homology	0/7	1.62	3.12	5.27	1/7	1.29	1.3	2.01	0/7	4.85	7.88	11.72	4/7	**0.62**	**0.89**	**1.44**
MiniBooNE Particle Identification	0/7	2.37	4.12	6.32	2/7	0.91	**0.93**	**1.37**	0/7	8.15	12.24	18.94	2/7	**0.71**	1.12	3.02
MiniBooNE Particle Identification (normalized)	0/7	0.31	0.8	1.29	0/7	4.83	6.04	7.44	2/7	0.81	1.2	1.63	4/7	**0.18**	**0.3**	**0.61**
MFCCs for Speech Emotion Recognition	0/7	0.26	0.74	1.28	0/7	0.62	0.71	1.04	3/7	0.73	0.98	1.29	3/7	**0.2**	**0.29**	**0.47**
ISOLET	0/7	0.98	3.31	4.75	3/7	0.36	0.38	**0.44**	1/7	0.41	0.63	1.32	3/7	**0.22**	**0.3**	0.46
Sensorless Drive Diagnosis	0/7	0.73	1.4	2.48	0/7	0.51	0.53	0.75	0/7	1.25	2.12	3.96	0/7	0.25	0.38	0.72
Sensorless Drive Diagnosis (normalized)	0/7	0.05	0.23	0.33	0/7	0.7	0.88	1.27	1/7	0.1	0.14	0.23	0/7	0.06	0.14	0.33
Online News Popularity	0/7	0.14	0.49	0.87	0/7	0.3	0.31	0.48	1/7	0.51	0.83	1.21	3/7	0.12	**0.18**	**0.29**
Gas Sensor Array Drift	0/7	0.42	1.3	2.09	5/7	0.16	**0.16**	**0.21**	1/7	0.25	0.61	1.29	1/7	**0.14**	0.27	0.48
3D Road Network	0/7	0.16	0.46	1.33	0/7	2.17	2.76	3.61	0/7	1.89	2.28	3.1	0/7	0.42	0.62	1.03
Skin Segmentation	0/7	0.03	0.16	0.21	0/7	1.05	1.06	1.56	0/7	0.06	0.08	0.11	0/7	0.06	0.15	0.26
KEGG Metabolic Relation Network (Directed)	0/7	0.27	0.8	1.15	0/7	0.34	0.36	0.89	0/7	1.2	1.61	2.08	0/7	0.27	0.43	**0.84**
Shuttle Control	0/8	0.26	0.96	1.47	0/8	0.28	0.31	0.49	0/8	0.09	0.17	0.35	0/8	0.18	0.26	0.46
Shuttle Control (normalized)	0/8	0.04	0.26	0.39	0/8	0.6	0.84	1.24	0/8	0.02	0.02	0.03	0/8	0.04	0.09	0.2
EEG Eye State	0/8	0.22	0.94	1.46	0/8	0.08	0.08	0.1	0/8	0.07	0.12	0.19	0/8	0.06	0.09	0.14
EEG Eye State (normalized)	0/8	0.17	0.71	0.96	0/8	0.83	1.26	1.97	0/8	0.06	0.11	0.21	0/8	0.05	0.09	0.17
Pla85900	0/7	0.07	0.9	1.48	0/7	0.34	0.35	0.62	0/7	0.05	0.08	0.13	0/7	0.09	0.16	0.29
D15112	0/7	0.15	1.0	1.44	0/7	0.07	0.09	0.09	4/7	**0.01**	**0.01**	**0.02**	0/7	0.07	0.11	0.16
Overall Results	4/165	2.0	4.13	6.0	15/165	8.45	8.82	9.56	19/165	6.31	8.12	11.53	35/165	**1.24**	**1.9**	**2.89**

Dataset	K-means++				CURE				CluDataSE				LW-coreset			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	0/7	464.35	815.42	1536.13	0/7	33.29	38.02	51.0	0/7	335.89	693.94	1360.43	0/7	23.66	33.47	52.41
HEPMASS	0/7	312.36	590.65	929.22	0/7	47.1	49.38	56.46	0/7	155.94	412.73	1263.65	5/7	6.34	**7.22**	**8.88**
US Census Data 1990	0/7	49.42	73.35	132.69	0/7	3.36	3.7	4.77	0/7	30.64	54.71	165.78	0/7	3.22	3.34	4.72
Gisette	0/7	39.19	63.33	105.25	0/7	23.87	25.3	34.47	0/7	51.86	82.55	136.48	0/7	23.28	33.97	53.87
Music Analysis	0/7	51.27	79.46	216.08	0/7	3.36	3.69	5.17	0/7	60.2	99.23	184.61	1/7	2.11	3.37	5.89
Protein Homology	1/7	7.86	14.41	26.7	0/7	87.88	93.78	107.19	0/7	20.36	26.57	37.94	1/7	3.12	4.35	10.36
MiniBooNE Particle Identification	1/7	3.33	7.96	14.53	0/7	217.68	243.64	272.31	0/7	11.9	15.23	18.55	2/7	0.99	1.82	3.18
MiniBooNE Particle Identification (normalized)	0/7	4.41	6.91	16.33	0/7	6.53	7.04	7.93	0/7	5.11	8.31	14.57	1/7	0.26	0.4	0.62
MFCCs for Speech Emotion Recognition	0/7	2.17	3.89	6.68	0/7	7.86	8.99	10.31	0/7	2.88	5.42	8.42	1/7	0.23	0.44	0.82
ISOLET	0/7	1.28	1.92	4.12	0/7	1.87	1.94	2.2	0/7	1.19	2.07	3.98	0/7	0.55	0.77	1.44
Sensorless Drive Diagnosis	0/7	0.82	1.42	2.77	0/7	4.5	4.82	5.62	0/7	1.73	2.69	4.71	7/7	**0.12**	**0.19**	**0.34**
Sensorless Drive Diagnosis (normalized)	0/7	0.43	0.81	2.09	0/7	1.18	1.35	1.59	0/7	0.34	0.78	1.84	6/7	**0.03**	**0.05**	**0.09**
Online News Popularity	0/7	0.52	0.79	1.92	0/7	6.16	6.73	7.77	0/7	1.45	2.05	4.19	3/7	**0.1**	**0.18**	0.43
Gas Sensor Array Drift	0/7	0.33	0.55	1.11	0/7	6.86	8.12	11.61	0/7	0.63	1.19	1.9	0/7	0.19	0.36	0.81
3D Road Network	0/7	1.36	5.39	9.37	0/7	14.51	16.58	19.05	0/7	10.66	11.14	12.9	7/7	**0.04**	**0.07**	**0.13**
Skin Segmentation	0/7	0.17	0.29	0.8	0/7	4.46	4.9	5.57	0/7	0.24	0.43	0.76	7/7	**0.01**	**0.02**	**0.03**
KEGG Metabolic Relation Network (Directed)	0/7	0.21	0.36	1.28	0/7	4.81	5.44	13.67	0/7	1.76	2.02	2.49	7/7	**0.05**	**0.08**	2.53
Shuttle Control	0/8	0.04	0.1	0.22	0/8	6.35	7.06	7.83	0/8	0.45	0.58	0.76	8/8	**0.01**	**0.02**	**0.04**
Shuttle Control (normalized)	0/8	0.05	0.09	0.16	0/8	0.57	0.6	0.72	0/8	0.04	0.07	0.15	8/8	**0.01**	**0.01**	**0.01**
EEG Eye State	1/8	0.05	0.09	0.24	0/8	3.12	3.44	4.13	0/8	0.9	0.96	1.1	7/8	**0.02**	**0.03**	**0.08**
EEG Eye State (normalized)	0/8	0.05	0.11	0.27	0/8	3.36	3.54	3.96	0/8	0.46	0.74	1.02	8/8	**0.02**	**0.04**	**0.08**
Pla85900	0/7	0.14	0.3	0.75	0/7	160.17	176.9	204.49	0/7	0.15	0.29	0.75	7/7	**0.02**	**0.03**	**0.08**
D15112	0/7	0.02	0.03	0.08	0/7	11.13	12.25	13.56	0/7	0.06	0.08	0.11	3/7	**0.01**	**0.01**	**0.02**
Overall Results	3/165	40.86	72.51	130.82	0/165	28.7	31.62	37.02	0/165	30.21	61.9	140.31	89/165	2.8	3.92	6.39

5.7.5 Analysis of obtained results

In terms of clustering accuracy ε, averaged across all the datasets, only the following three algorithms exhibited satisfactory results: Big-means, K-means++, and LW-coreset. Notably, Big-means was the top performer, showing a substantial lead in overall effectiveness. In fact, it was the only algorithm that achieved the average accuracy gap of less than 1%. The remaining algorithms yielded less consistent outcomes, particularly when applied to specific datasets, such as "MiniBooNE Particle Identification" and "EEG Eye State" datasets. These datasets are marked by an elevated susceptibility to clustering instability, where even minor deviations in the initial centroid positions yield significantly deteriorated final optima.

Big-means attained the best accuracy results on the small and large datasets, while achieving better or comparable accuracy results for the biggest datasets. The "Max" statistic of Big-means has the lowest values across the majority of datasets, and this reveals that Big-means possesses the highest stability with respect to a large number of executions. We attribute this property to the iterative sampling nature of Big-means, especially to its shaking effect that allows Big-means to avoid the valleys of bad local minima.

Regarding the clustering time t, the K-means++, CURE, CluDataSE and BDCSM algorithms exhibited the slowest performance. By looking at the obtained time scores of these algorithms for the biggest datasets in the list, one can easily note the inability of these algorithms to efficiently cluster datasets with a huge number of entries. The Minibatch K-means and LW-coreset were the fastest on average, with Minibatch K-means leading in the big datasets, and LW-coreset leading in the small and large datasets.

Adopting the trade-off between the clustering accuracy and time as a criterion, the Big-means algorithm appears as an obvious leader. Big-means achieved the top average accuracy score, while exhibiting the best or comparable time results among the most accurate algorithms. It is interesting to note that although Minibatch K-means was the fastest algorithm by far, it also produced one of the most inaccurate average results. This observation justifies our belief that although simplicity of an algorithm is crucial for the ease of implementation,

parallelization, and processing speed, excessive simplicity can undermine the algorithm's accuracy and effectiveness.

Thus, in developing new algorithms, it is vital to strike the right balance between the aspects of effectiveness (accuracy), efficiency (time), and simplicity (implementation and time). As we mentioned earlier, the LIMA dominance criterion [Bri+23] is a measure that takes into account all these aspects. In this study, we adopt LIMA dominance as the final measure of the obtained experimental results. Among the algorithms that obtained the best accuracy scores (Big-means, K-means++, and LW-coreset), Big-means stands out as LIMA-dominant:

$$\text{Big-means}(0.6, 4.13, 6) \underset{LIMA}{>} \text{K-means++}(4.15, 72.51, 6)$$
$$\text{Big-means}(0.6, 4.13, 6) \underset{LIMA}{>} \text{LW-coreset}(39.39, 3.92, 7)$$

Here the notation $A(accuracy,\ time,\ LIMA\ number)$ is used to represent the performance of an algorithm, and the last LIMA comparison was made assuming that the time results of 4.13 and 3.92 seconds were comparable.

5.8 Conclusion and future research

The findings of our study in this chapter highlight the challenges involved in solving the NP-hard MSSC problem in big data conditions using K-means-like methods. Based on the comprehensive review of the available approaches for optimizing K-means, it is apparent that only very few ones can be applied to real big data. In our experimental findings, Big-means [Mus+23] emerged as a standout big data clustering algorithm. It outperformed traditional and other tested MSSC methods at all data sizes (small, large, and big), achieving a perfect balance of accuracy, speed, and simplicity, as proven by the LIMA dominance criterion [Bri+23]. The inherent simplicity of Big-means challenges the notion that more intricate hybrid methodologies are required to achieve superior clustering outcomes. While Big-means does necessitate a well-informed selection for its sample size parameter s, it has

the potential to serve as a definitive algorithm within the realm of MSSC algorithms based on K-means [Mus+23]. Nevertheless, its efficacy on big data could be further enhanced through the integration of advanced metaheuristics.

The LW-coreset algorithm for coreset construction, which necessitates a minimum of two passes over the complete dataset, proves infeasible under big data constraints. Similarly, many alternative approaches either demand multiple dataset passes or utilize intricate concepts and metaheuristics, resulting in high computational demands. These methods often grapple with memory overflow issues or exorbitant processing times [Mus+23]. While sampling-based clustering techniques such as BDCSM, Minibatch K-means, and CluDataSE expedite processing time through sampling, their final clustering accuracy remains suboptimal. Singularly applying other methods intrinsic to big data clustering algorithms, like parallelization, triangle inequality, employing streamlined data structures, or hybridization, fails to yield satisfactory outcomes on big datasets. We posit that these strategies truly shine when integrated as enhancements to a big data clustering algorithm rooted in foundational principles like the "less is more" and decomposition approaches [Mus+23]. Such principles are quintessential for genuine big data clustering algorithms.

Other recent surveys of scalable and parallel big data clustering methods emphasize that most of the existing algorithms have high-complexity implementations. Furthermore, few existing methods have shown support for the velocity and variety characteristics of big data. Most parallel clustering algorithms proposed in the literature do not handle real-time data and focus on a single, mostly numerical, type of data.

In this study, based on our research findings and critical insights, we provide guidelines for selecting the optimal big data clustering algorithm. We anticipate that the accompanying infographic and the algorithm selection flowchart will serve as useful tools for practitioners struggling with emergent complexities inherent to big data clustering.

Our research underscores the pressing need for innovative MSSC algorithms adept at addressing the NP-hard problem in the context of big data. These solutions should be scalable, efficient, and streamlined to maximize their practical relevance. They ought to

transcend the limitations inherent to current methodologies, such as scalability issues, the demand for multiple dataset passes, memory limitations, and the constraints of processing real-time data across multiple modalities. Upcoming research endeavors should prioritize the design of these novel algorithms, while also benchmarking them against existing approaches like the Big-means algorithm.

With the continuous advancement in computational power and data generation, the landscape of big data clustering continues to evolve. Techniques such as online clustering, where the algorithm adapts to data arriving in a stream, and ensemble (consensus) clustering, which combines multiple clustering results, present exciting avenues for future research and development. Additionally, we see untapped potential in the Big-means algorithm, which could be further enhanced by integrating modern metaheuristics like Variable Neighborhood Search (VNS), opening up new possibilities for improvement [MH97]. We also aim to develop methods for automatically selecting the best parameters for big data clustering algorithms, including sample size and parallelization strategy, to optimize performance. Finding ways to combine state-of-the-art deep learning techniques with big data clustering approaches is another exiting future research direction.

Chapter 6

VNS-ACCELERATED GLOBAL OPTIMIZATION OF SUM-OF-SQUARES CLUSTERING FOR BIG DATA

6.1 Introduction

6.1.1 Motivation

Central to the success of Big-means is the decomposition principle, which constructs a solution to the MSSC task (1.1) from solutions to its subproblems. The goal of each subproblem is to solve the MSSC task approximately, on a random sample from the original dataset. By allowing the current solution to flow between the approximate MSSC solutions obtained on the drawn samples, Big-means is able to explore the global solution space more effectively and efficiently than other available approaches. Big-means organizes this flow according to the "keep the best" approach. This not only forms the foundation of the algorithm but also enables its efficient and easy parallelization, which is a key strategy in the field of big data.

Nevertheless, there exist more advanced ways to combining partial solutions into a global one. In particular, the set of all partial solutions can be endowed with some special structure. Exploring it in accordance to some set of rules may give enough guidance to achieve a more educated traversal of partial solutions. This guidance can be provided by a suitable metaheuristic, such as Variable Neighborhood Search (VNS) [MH97], simulated annealing, tabu search, genetic algorithms, branch and bound, etc.

Now, we endeavor to optimize the Big-means algorithm along the exiting dimension of applying metaheuristic ideas to the decomposition process. Chapter 3 did not delve into this essential aspect, leaving room for further exploration. In this chapter, we aim to explore this dimension in detail, providing practical insights into the hybridization of big data clustering with a modern optimization metaheuristic.

This chapter introduces a novel big data clustering technique named BigVNSClust, which seeks to amplify the global optimization capabilities of the Big-means algorithm. By integrating a powerful metaheuristic framework, Variable Neighborhood Search (VNS) [MH97], we develop a refined approach to clustering. A comprehensive experimental analysis reveals that BigVNSClust demonstrates state-of-the-art performance within the MSSC clustering model. Notably, it surpasses its predecessor, Big-means, solidifying its position as a robust and innovative advancement in the field.

6.1.2 Key results

One of the anticipated drawbacks of K-means++ is its inability to obtain a good quality solution when two or more initial centroids fall inside a single cluster that is well-separated from the other data points by a large margin. This occurs due to inability of centroids to travel across the empty space of the margin by shifting centroids to the means, even if the dataset is sparsified well enough by its small sample. This issue was empirically demonstrated in Section 3.7 of Chapter 3.

The BigVNSClust algorithm builds upon Big-means, while employing a novel metaheuristic idea borrowed from the VNS framework [MH97; Bri+23]. More concretely, in the intermediate iterations of the algorithm, not only degenerate clusters but also p random centroids are reinitialized using K-means++. Variable p is called the shaking power. It is cyclically incremented across iterations, being bounded by the hyperparameter $p_{\max}$.

This improvement helps BigVNSClust effectively tackle the situation when multiple centroids of the current solution are stuck inside one cluster, enhancing the overall accuracy and efficiency of the algorithm. We confirm this result by extensive computational experiments on real-world datasets.

6.1.3 Chapter structure

The chapter has the following outline. Section 6.2 provides an elementary introduction to the VNS metaheuristic. Section 6.3 gives a precise pseudocode of BigVNSClust, discusses

its main properties, as well as analyzes the resulting time complexity. Section 6.4 describes the conducted experiments and analyzes the obtained results. Finally, Section 6.5 concludes the chapter with final thoughts.

6.2 Variable Neighborhood Search

6.2.1 Main concepts

In the realm of computer science, artificial intelligence, and mathematical optimization, heuristics represent essential tools designed to expedite problem solving. They are used when conventional methods prove too slow or when the quest for an exact solution becomes intractable. It is crucial to acknowledge, however, that heuristics do not provide an absolute guarantee of uncovering the optimal solution, often falling under the category of approximate algorithms. Typically, these algorithms excel at swiftly and efficiently generating solutions that closely approximate the optimal one. On occasion, they may even attain highest accuracy in identifying the absolute best solution. Nevertheless, they retain their heuristic classification until the solutions produced by them are proved to be optimal [Des+15]. In a broader context, metaheuristics serve as versatile frameworks for crafting heuristics to tackle a diverse array of combinatorial and global optimization challenges [HM14].

An optimization problem can be generally formulated as

$$\min\{f(x) \mid x \in S \subseteq \mathcal{S}\}, \tag{6.1}$$

where:

- $\mathcal{S}$ represents the ambient solution space;
- S denotes the feasible set within $\mathcal{S}$;
- x is a feasible solution in S;
- f is a real-valued objective function.

Henceforth, the symbol S is used to represent two distinct concepts: firstly, a feasible solution set S within the ambient solution space $\mathcal{S}$; and secondly, a specific sample S extracted from the entire dataset X. The particular interpretation of S in any given instance should be readily discernible to the reader based on the surrounding context since the feasible solution space is $S = \mathcal{S} = \mathbb{R}^n$ in the context of MSSC (1.1).

Variable Neighborhood Search (VNS), introduced by Mladenovic in 1997 [MH97], represents a contemporary metaheuristic approach offering a versatile framework for effectively addressing combinatorial and continuous non-linear global optimization problems. VNS systematically exploits the idea of neighborhood change, both in descent to local minima and in escape from the valleys containing them [MH97; HM03; HM14]. It explores distant neighborhoods of the current solution and moves to a new one if and only if this movement leads to an improvement in the objective function.

VNS (Variable Neighborhood Search) stands as a potent instrument for the construction of novel global search heuristics, leveraging already existing local search heuristics. Its efficacy is underpinned by the following key insights:

Fact 1: A solution that constitutes a local optimum within one neighborhood structure does not necessarily remain optimal when analyzed under a different neighborhood;

Fact 2: A solution that achieves global optimality will simultaneously be a local optimum across all conceivable neighborhood structures;

Fact 3: For a broad spectrum of problems, local optima, when observed in the context of one or multiple neighborhoods, often lie in close proximity to one another.

Fact 1 paves the way for the adoption of intricate moves that aim to discern local optima across all the neighborhood structures in play. Fact 2 implies that expanding the number of neighborhoods might be a strategic move, especially if the current local optima identified are not of satisfactory quality. When combined, these insights highlight an intriguing prospect:

a solution recognized as a local optimum across multiple neighborhood structures has a heightened likelihood of achieving global optimality compared to one that is optimal within a single neighborhood structure [HMT+16].

Fact 3, primarily rooted in empirical findings, indicates that a local optimum can often shed light on the nature of the global optimum. Put differently, local optima in optimization problems tend to exhibit similarities among themselves. This consistency underscores the merit of intensively exploring the vicinity of a given solution. To elucidate further, if we collect all local solutions $C = (c_1, \ldots, c_k)$ of the MSSC problem as described in equation (1.1) into a single set, then it is plausible that a substantial subset of these locally optimal centroids will be very close to each other. In the meantime, a few might deviate. This pattern hints that the global optimum will bear resemblances in certain variables (specific coordinates of the vector representations) with those found in local optima. Identifying these overlapping variables beforehand is typically elusive. As a result, a systematic exploration of the neighborhoods surrounding the current local optimum is prudent until a superior solution emerges.

VNS comprises two phases: the improvement phase, which refines the current solution by potentially descending to the nearest local optimum, and the shaking (or perturbation) phase, designed to mitigate local minima traps. The VNS operates by alternately executing the improvement and shaking procedures, along with a neighborhood change step, until certain predefined stopping criteria are met. Specifically, VNS revolves around three core, alternating steps:

1. Shaking procedure;

2. Improvement procedure;

3. Neighborhood change step.

Now, let us define some essential notions for the VNS framework.

The incumbent solution is referred to as the best obtained solution x with respect to the objective function value.

A neighborhood of a given solution x refers to the set of solutions that can be directly obtained from x by applying a specific local change. Usually, a neighborhood is constructed based on a metric (or quasi-metric) function denoted as δ. Given a non-negative distance value $\Delta \geq 0$, the neighborhood of a point x can be defined as:

$$\mathcal{N}_\Delta(x) = \{y \in S \mid \delta(x, y) \leq \Delta\} \tag{6.2}$$

For instance, in the context of a continuous optimization problem over $\mathbb{R}^n$, $\mathcal{N}_1(x)$ may represent an Euclidean ball of radius 1 centered at x. Alternatively, $\mathcal{N}_3(x)$ could denote the set of all vectors derived from x by swapping (or changing) exactly three coordinates of x. In the Traveling Salesman Problem (TSP), one type of neighborhood might consist of all the tours that can be obtained by reversing a subsequence of a given tour. This specific operation is called a "2-opt" move.

A neighborhood structure $\mathcal{N}$ refers to an ordered collection of operators

$$\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_{k_{\max}}\},$$

where each operator $\mathcal{N}_k : S \to 2^S$ (2^S denotes the power set of S), for $k \in \{1, \ldots, k_{\max}\}$, maps a solution x to the predefined neighborhood $\mathcal{N}_k(x)$. The hierarchy of these neighborhoods plays a pivotal role in VNS. When a local search within one neighborhood fails to find any improvement, VNS typically progresses to the subsequent neighborhood in the sequence, aiming to discover superior solutions. It is common for VNS to employ multiple neighborhood structures throughout its search process, adjusting the neighborhood dynamically during the search to sidestep local optima. In the ensuing discussions, both the collection of operators $\mathcal{N}$ and the collection of neighborhoods $\mathcal{N}(x)$ will be referred to as a neighborhood structure.

Each neighborhood structure typically has a different way of defining neighbors. For instance, in the context of the TSP as previously discussed, one could utilize a neighborhood

structure based on "k-opt" moves. In a "k-opt" move, k edges are removed from the tour and subsequently reconnected in a manner different from the original configuration.

We adopt two distinct notations, $\mathcal{N}$ for the neighborhood structures in the shaking procedure and N for those in the improvement procedure, to differentiate between them.

6.2.2 Shaking procedure

A shaking procedure is a necessary step that allows to escape local optima traps.

The most straightforward shaking procedure involves selecting a random solution from the neighborhood $\mathcal{N}_k(x)$, with k being predetermined. For certain problems, a completely random jump within the k-th neighborhood may result in an overly aggressive perturbation. As a consequence, an "intensified shaking" approach is sometimes favored. This method considers the sensitivity of the objective function to minor variations in the variable x. Nevertheless, for the purposes of this chapter, our definition of the shaking procedure will align with the simpler, random-selection approach. The corresponding pseudocode for this shaking procedure is provided in Algorithm 20.

Algorithm 20: Shaking Procedure

```
1 Function Shake(x, k, 𝒩, P):
      Input  : Incumbent solution x;
               Neighborhood index k;
               Neighborhood structure 𝒩;
               Probability distribution P;
      Output: Shaken solution x'.
2     Choose x' ∈ 𝒩_k(x) at random according to P;
3     return x'
```

6.2.3 Neighborhood change step

The neighborhood change procedure aims to steer the direction of the variable neighborhood search heuristic throughout the solution space. Specifically, it dictates which neighborhood

will be probed next and determines whether a given solution should be adopted as the new incumbent. While various versions of the neighborhood change step can be found in the literature [HMT+16], the sequential and cyclic methods are the most prevalent.

1. Sequential neighborhood change. The steps of this procedure are listed in Algorithm 21. If an improvement of the incumbent solution in some neighborhood occurs, then the search is resumed in the first neighborhood (according to the predefined order in $\mathcal{N}$) of the neighborhood structure at the new incumbent solution; otherwise, the search is continued in the subsequent neighborhood of the incumbent solution.

Algorithm 21: Sequential Neighborhood Change Step

```
1 Function Neighborhood_change_sequential(x, x', k):
      Input  : Incumbent solution x;
               Candidate solution x';
               Neighborhood index k.
2     if f(x') < f(x) then
3         x ← x'; // move
4         k ← 1; // return to first neighborhood
5     else
6         k ← k + 1; // proceed to next neighborhood
7     end
```

2. Cyclic neighborhood change. In this step, regardless of whether there has been an improvement with respect to some neighborhood or not, the search is continued in the subsequent neighborhood (see Algorithm 22):

6.2.4 Improvement procedures

There are two main types of improvement procedures that can be used inside a VNS heuristic: local search and variable neighborhood descent (VND) [HMT+16].

A local search heuristic offers a fundamental approach to solution improvement. At each iteration, it delves into the neighborhood structure $N(x)$ of the incumbent solution x. The

Algorithm 22: Cyclic Neigborhood Change Step

```
1 Function Neighborhood_change_cyclic(x, x′, k):
      Input  : Incumbent solution x;
               Candidate solution x′;
               Neighborhood index k.
2     if f(x′) < f(x) then
3         x ← x′; // move
4     end
5     k ← k + 1; // proceed to next neighborhood
```

process starts with this incumbent solution. If a superior solution within the predefined neighborhood of $N(x)$ is identified, it then replaces the incumbent. This iterative refinement continues until reaching a solution that is locally optimal with respect to its neighborhood, signifying no further enhancements are possible.

Two prevalent strategies to navigate the neighborhood $N(x)$ are:

1. First Improvement: This strategy immediately adopts the first detected improved solution within $N(x)$ as the new incumbent;

2. Best Improvement: Here, the most optimal solution amongst all potential improvements in $N(x)$ becomes the new incumbent.

For the scope of this work, our primary focus is on the local search using the best improvement strategy. The corresponding pseudocode is detailed in Algorithm 23.

A Variable Neighborhood Descent (VND) procedure delves into multiple neighborhoods of the incumbent solution in every iteration. By operating in either a sequential or nested manner, its objective is to refine the current solution. In essence, VND capitalizes on the initial two VNS characteristics previously mentioned. While the choice between the "first improvement" and the "best improvement" search strategies remains flexible in VND, the basic sequential VND (B-VND) procedure expands the concept of local search. In this context, if a neighborhood structure encompasses several neighborhoods denoted by $N =$

Algorithm 23: Local Search using Best Improvement

```
1 Function Best_improvement_local_search(x, N):
      Input  : Incumbent solution x;
               Neighborhood N;
      Output: New incumbent solution x'.
2     repeat
3         x' ← x; // remember old solution
4         x ← arg min_{y∈N(x')} f(y); // find best improving solution
5     until f(x') ≤ f(x);
6     return x'
```

$\{N_1, \ldots, N_{l_{\max}}\}$ (with $l_{\max}$ setting the upper limit on the number of neighborhoods), B-VND systematically investigates these neighborhoods based on this predetermined sequence. When an enhancement in the incumbent solution emerges within a particular neighborhood, the search reinitiates from the foremost neighborhood but pivots around this newly improved solution. The entire procedure culminates when no further refinements can be achieved for the incumbent solution across all $l_{\max}$ neighborhoods. The procedural steps of the sequential VND, adopting the best improvement search strategy, are detailed in Algorithm 24.

Also, there is a plethora of other VND procedures: pipe VND (P-VND), cyclic VND (C-VND), union VND (U-VND), nested VND, mixed VND, etc. [HMT+16] However, since they were not used in the algorithm proposed in this chapter, we omit their description and refer the reader to [HMT+16] for more details.

6.2.5 Basic Variable Neighborhood Search

Unlike many other metaheuristics, Variable Neighborhood Search (VNS) and its various extensions are characterized by their simplicity, often requiring minimal or even zero tuning of parameters. Consequently, besides delivering high-quality solutions, often in more straightforward fashion compared to alternative approaches, VNS also offers insights into the underlying reasons for such effectiveness. These insights can, in turn, pave the way for the development of more efficient and refined implementations [HM14].

Algorithm 24: Sequential VND using Best Improvement Search

```
1  Function B_VND(x, l_max, N):
       Input  : Incumbent solution x;
                Maximum neighborhood index l_max;
                Neighborhood structure N;
       Output: New incumbent solution x'.
2      repeat
3          stop ← False;
4          l ← 1;
5          x' ← x;
6          repeat
7              x'' ← arg min_{y∈N_l(x)} f(y); // find best improving solution
8              Neighborhood_change(x, x'', l); // any neighborhood change function
9          until l = l_max;
10         if f(x') ≤ f(x) then
11             stop ← True;
12         end
13     until stop = True;
14     return x'
```

The pseudocode of the basic VNS is presented in Algorithm 25, while Figure 6.1 schematically shows the process of VNS optimization applied to some non-convex function f.

Within the basic VNS framework, the following key components can be identified:

- $f(x)$: Real-valued objective function;
- k: Shaking intensity;
- $k_{\max}$: Maximum shaking intensity;
- $\mathcal{N}_k(x)$: The k-th neighborhood of solution x;
- x: Incumbent (current) solution;
- $x' \leftarrow \text{Shake}(x, k, \mathcal{N})$: A procedure, detailed in Section 6.2.2, that generates a candidate

Algorithm 25: Basic Variable Neighborhood Search

```
1  Function Basic_VNS(x, k_max, T, 𝒩, N):
       Input  : Initial solution x;
                Maximum shaking power k_max;
                Maximum time T;
                Neighborhood structure for shaking 𝒩;
                Neighborhood structure for improvement N;
       Output: Final solution x.
2      repeat
3          k ← 1 ; // initially, use the minimal shaking power
4          while k ≤ k_max do
5              x' ← Shake(x, k, 𝒩); // generate k-th power variation of incumbent
                solution
6              x'' ← Local_search(x', N); // apply some local search method with x' as
                initial solution
7              Neighborhood_change(x, x'', k); // apply one of described neighborhood
                change methods
8          end
9      until CpuTime() > T;
10     return x
```

solution x' randomly selected from the k-th neighborhood $\mathcal{N}_k(x)$ of the incumbent solution x;

- NeighborhoodChange(x, x'', k): A procedure for progressively altering the neighborhood, as described in Section 6.2.3;
- $x'' \leftarrow$ LocalSearch(x', N): A local search method, elaborated upon in Section 6.2.4, utilized to find solutions within the vicinity of local optima;
- CpuTime(): Function that returns the time elapsed since the algorithm's initiation;
- T: A predefined, relatively short time limit for the search.

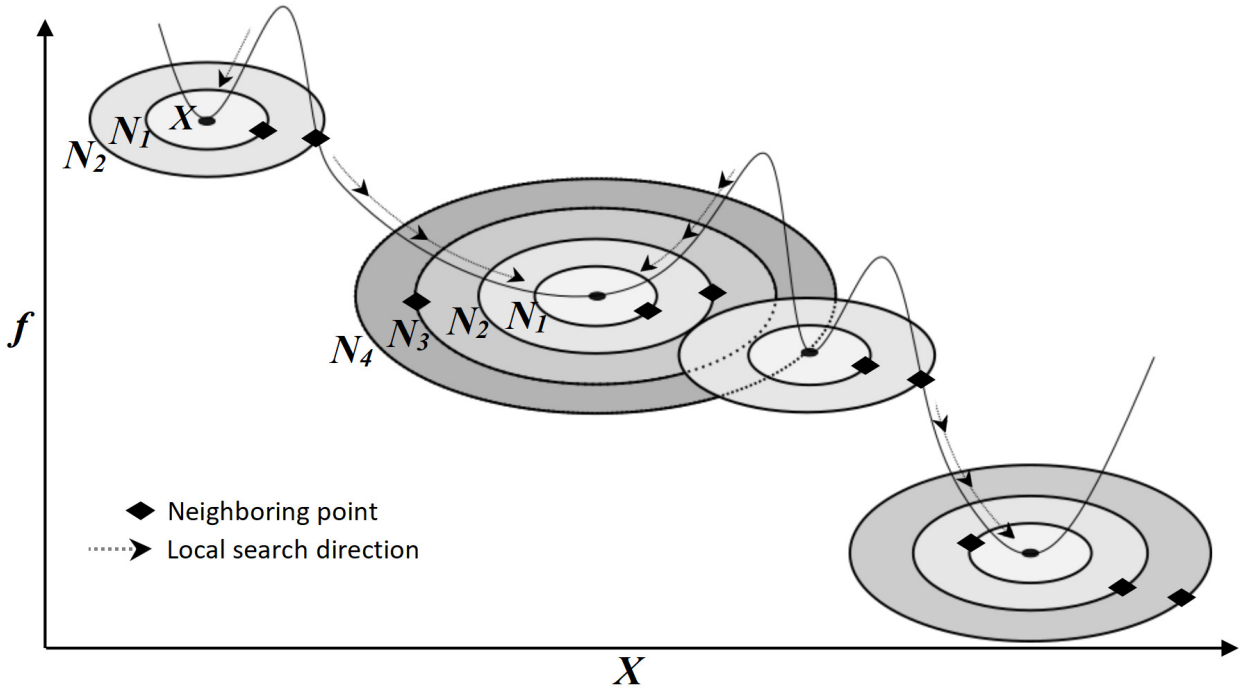


Figure 6.1: Illustration of VNS applied to minimize some function f. Reproduced from "Constructing the Neighborhood Structure of VNS Based on Binomial Distribution for Solving QUBO Problems" [PK22] by Pambudi and Kawamura, 2022, Algorithms, CC BY 4.0. https://creativecommons.org/licenses/by/4.0/

6.3 Proposed algorithm

6.3.1 Precise description

The pseudocode of the proposed BigVNSClust algorithm is shown in Algorithm 26. In the context of MSSC, where k already denotes the number of clusters, we use p and $p_{\max}$ to represent the current and maximum shaking powers, respectively.

The proposed Algorithm 26 follows the basic VNS variation, as described in Algorithm 25, using the following choice of ingredients:

1. Neighborhood structure $\mathcal{N}$ employed by the shaking procedure is defined as follows. Let $S \subset X$ be a uniform random sample of size s from X. Also, let $U = (u_1, \ldots, u_l) \in \mathbb{R}^{l \times n}$ be a finite set of points in X. Define a probability distribution $P : X \to [0, 1]$ by

$$P_U(x) = \frac{d(x, U)}{\sum_{x' \in X} d(x', U)} \tag{6.3}$$

where $d(x, U)$ denotes the distance from point x to set U. The distribution P is exactly the distribution used in the K-means++ seeding. Let $C = (c_1, \ldots, c_k)$ be the incumbent solution. Then, the p-th neighborhood $\mathcal{N}_p(C)$ of incumbent solution C is defined as follows:

$$\begin{aligned} \mathcal{N}_p(C) = \{(c_1, c_2, \ldots, c_{l_1}, \ldots, c_{l_2}, \ldots, c_{l_p}, \ldots, c_k) \mid \\ \text{where } l_1, l_2, \ldots, l_p \in \mathbb{N}_k \text{ such that } l_i \neq l_j \text{ for } i \neq j, \\ \text{and for every } i \in \mathbb{N}_p,\ U_i = \{c_j\}_{j \in \mathbb{N}_k, j \notin \{l_i, \ldots, l_p\}}, \\ c_{l_i} \in S \setminus U_i \text{ is sampled according to } P_{U_i}\}; \end{aligned} \tag{6.4}$$

2. The local search procedure follows Algorithm 23. The neighborhood structure $N(C)$ for incumbent solution C is defined as

$$N(C) = \bigcup_{i \in \mathbb{N}_k} \text{Conv}(S_i),$$

Algorithm 26: Big Data Clustering Using VNS

Function `BigVNSClust`(X, k, s, $p_{\max} = 3$)**:**
  Input : Feature vectors $X = \{x_1, \ldots, x_m\}$;
    Desired number of cluster centers k;
    Sample size s;
    Maximal shaking power $p_{\max}$ with default value = 3;
  Output: Cluster centers $C = \{c_1, \ldots, c_k\}$;
    Point-to-cluster assignments $Y = \{y_1, \ldots, y_m\}$;
    Objective function value $f(C, X)$.
  $C \leftarrow \{\varnothing_1, \ldots, \varnothing_k\}$; `// initially, every centroid is degenerate`
  $\hat{f} \leftarrow \infty$;
  $p \leftarrow 1$;
  Set iteration counter $t = 0$;
  while $t < T$ **do**
    Draw a random sample S of size s from X;
    Draw a random sample C' of p centroids from C;
    for *each centroid c in C* **do**
      if *c is in C' or is the empty set* **then**
        Reinitialize c using K-means++ on S;
      end
    end
    Compute new centroids C_{new} using K-means on S with initial centroids C;
    if $f(C_{new}, S) < \hat{f}$ **then**
      $C \leftarrow C_{\text{new}}$;
      $\hat{f} \leftarrow f(C_{\text{new}}, S)$;
    end
    $p \leftarrow p + 1$;
    if $p > p_{\max}$ **then**
      $p \leftarrow 1$
    end
    $t \leftarrow t + 1$;
  end
  $Y \leftarrow$ Assign each point in X to nearest centroid in C;
  return $C, \quad Y, \quad f(C, X)$

where $S_i \subset S,\ i \in \mathbb{N}_k$, stands for the set of all sample points belonging to cluster c_i, i.e., $S_i = \{x \in S \mid \arg\min_{c \in C} d(x, c) = c_i\}$, while $\text{Conv}(\cdot)$ is the operation of taking the convex hull of a set. For each cluster, searching for the best improving candidate solution inside the convex hull of all points inside the cluster is justified due to an important property of the squared Euclidean distance function. If a centroid is placed outside the convex hull of the points, the objective function is guaranteed to decrease; whereas, the closer the centroid is placed to the mean of cluster points, the smaller the squared Euclidean distance from the points to the centroid becomes [CYY20]. It is not hard to show that the maximum improvement of the objective function (1.1) is attained by placing new centroids at the means of the clusters [CYY20]. K-means is based exactly on this crucial property. Hence, K-means is used as the local search procedure in the proposed algorithm;

3. The neighborhood change step follows Algorithm 22. In each iteration, the shaking power p is incremented irrespective of whether the objective function has decreased or not.

It has been established empirically that it is exactly the above choice of VNS ingredients that guarantees the best performance of BigVNSClust both with respect to the resulting time and quality of the obtained solution.

It should be noted that BigVNSClust does not strictly adhere to the VNS metaheuristic, as both the neighborhood structure $\mathcal{N}$ for shaking and the local search procedure are applied to a sample S rather than the entire data space X. Nonetheless, Chapter 8 introduces a new metaheuristic, Variable Landscape Search (VLS), under which BigVNSClust will be categorically included.

6.3.2 Analysis of the algorithm

In contrast with the classical Big-means scheme (Section 3.4), BigVNSClust employs a new explicit shaking procedure. After the local search phase, exactly p centroids, along with

any degenerate ones, are reinitialized in the incumbent solution C. Clearly, as p increases, the perturbation applied to C becomes more intense. In the edge case when $p_{\max} = k$ and p reaches $p_{\max}$, the whole incumbent solution is reinitialized according to the K-means++ logic. This amounts to a total restart. However, it is most practical to set $p_{\max}$ much smaller than k. This limits the strength of applied perturbations, and therefore helps to avoid a complete or near-complete restart.

The shaking power p is increased in every iteration. This allows to incrementally, but in a limited manner, expand the search space for each new incumbent solution in the local search procedure. By employing the K-means++ logic to sample new centroids in the shaken incumbent solution, we ensure that the new centroids are distributed throughout the sample for optimal coverage. This strategy mitigates the risk of the incumbent solution settling into local minima where multiple cluster centers are positioned so closely that they incorrectly segment a single true cluster.

Apart from the aforementioned differences, BigVNSClust is identical to the Big-means algorithm.

6.3.3 Time complexity

In Algorithm 26, the K-means++ reinitialization of p and all degenerate clusters (as seen in line 9) shares the same time complexity, $\mathcal{O}(s \cdot n \cdot k)$, with a single iteration of the K-means local search. This equivalence arises since all operations are executed on a sample S of size s. The additional segment in lines 19 to 22 of Algorithm 26, which deals with the incrementation of shaking power p, operates at a complexity of $\mathcal{O}(1)$. Consequently, the overall time complexity of a single iteration of the BigVNSClust method remains $\mathcal{O}(s \cdot n \cdot k)$, consistent with the preceding Big-means algorithm.

6.4 Experimental evaluation

For the information on the used datasets, software and hardware, experimental methodology, evaluation metrics, as well as other implementation details, we refer the reader to Section 4.6

of Chapter 4. In this chapter, the design of the experiments is identical to Chapter 4, except for some minor differences that we will describe below.

6.4.1 Range of algorithms

In this section, we conduct a detailed experimental analysis to quantify the enhancements brought by integrating the Variable Neighborhood Search (VNS) scheme into our novel BigVNSClust approach, in comparison with the standard Big-means algorithm. Specifically, our focus is to isolate and measure the impact of the added VNS component. To ensure a fair and focused comparison, we evaluate both algorithms under the same conditions, employing the inner parallelization technique, which was used in Chapter 3. This approach allows us to precisely attribute any observed improvements in performance to the inclusion of the VNS scheme.

Investigating the optimal parallelization approach for the BigVNSClust algorithm might be a subject for future research.

6.4.2 Time metric

For both algorithms, the average CPU time, denoted as $\bar{t}$, corresponds to the time required for convergence to the baseline results, as outlined in Section 4.6. The rationale for preferring this metric over the straightforward total clustering time is detailed in Section 4.6. Moreover, using the same time metric $\bar{t}$ as in Chapter 4 ensures an additional cross-comparison with the results presented in that chapter. The values of $\bar{t}$ are expressed in seconds.

6.4.3 Other implementation details

For a fair comparison, BigVNSClust was configured with the same values of parameters s and T that were used in the experiments of Chapter 3.

6.4.4 Experimental results

The total number of conducted experiments reached 7,366.

In our experimentation, we did not detect significant correlation between the obtained results and the choice of a neighborhood change procedure. However, the cyclic neighborhood change procedure produced slightly better final accuracy ε than the sequential one.

Also, we explored two distinct methodologies for centroid shaking within our algorithm: a uniformly random approach and a reinitialization based on the K-means++ strategy. It was observed that, while the uniformly random approach can be executed more rapidly than its K-means++ counterpart, it resulted in a marked decrease in final accuracy, measured by ε. Specifically, the accuracy deteriorated by up to fivefold in comparison to the K-means++ reinitialization. This significant discrepancy in performance can be attributed to the nature of the uniformly random method, which introduces an excessive level of perturbation to the existing solution. Such a drastic alteration necessitates a considerably higher number of iterations in the subsequent local search phase to converge to an optimal solution.

For every algorithm, dataset X, and number of clusters k, the minimum, median, and maximum values of relative accuracy ε and CPU time $\bar{t}$ were calculated with respect to n_{exec} runs. To determine the optimal value of the maximum shaking power $p_{\max}$, all the experiments were restarted with different values of $p_{\max}$. The resulting performance of the proposed BigVNSClust algorithm across various values of $p_{\max}$ is shown in Table 6.1.

Table 6.1: Summarized performance of the proposed BigVNSClust algorithm with varying values of the maximum shaking power $p_{\max}$

$p_{\max}$	Relative accuracy ε				Time $\bar{t}$			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max
2	114/165	0.12	0.78	**4.51**	4/165	1.07	**1.86**	**3.31**
3	112/165	**0.08**	0.72	1784.89	4/165	1.27	2.14	3.69
4	**122/165**	0.09	**0.67**	3000.59	4/165	1.0	2.11	3.57
5	108/165	0.1	0.68	5.67	3/165	**0.96**	2.03	3.72

Table 6.1 indicates that more intensive shaking leads to improved accuracy and a slightly increased convergence time. The optimal value of $p_{\max}$, in terms of median relative accuracy, is 4. From these results, it can be inferred that more vigorous shaking enhances the incumbent solution's ability to escape local minima by "jumping out" of their valleys. However, as anticipated, descending from these perturbed positions using local search incurs additional time.

For the best-performing configuration of BigVNSClust with parameter value $p_{\max} = 4$, the overall results are shown in Tables 6.2 – 6.3.

Table 6.2: Relative clustering accuracies ϵ (in %) resulting from the comparison of BigVNSClust with Big-means

Dataset	BigVNSClust				Big-means			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	4/7	**0.04**	**0.12**	**0.26**	3/7	0.06	0.27	0.44
HEPMASS	5/7	**0.07**	**0.14**	**0.26**	2/7	0.09	0.27	1.05
US Census Data 1990	7/7	**0.4**	**1.39**	**2.76**	0/7	0.89	3.17	33.23
Gisette	2/7	**-0.44**	-0.36	**-0.25**	5/7	**-0.44**	**-0.38**	-0.15
Music Analysis	4/7	0.34	**0.91**	**2.23**	3/7	**0.32**	1.16	7.48
Protein Homology	4/7	0.38	**0.83**	**1.97**	3/7	**0.14**	0.87	5.31
MiniBooNE Particle Identification	5/7	**-0.08**	**0.06**	**3.07**	2/7	-0.02	0.63	22.53
MiniBooNE Particle Identification (normalized)	3/7	0.28	0.68	**1.57**	4/7	**0.16**	**0.54**	3.23
MFCCs for Speech Emotion Recognition	5/7	**0.15**	**0.42**	**0.96**	2/7	0.19	0.87	1.43
ISOLET	5/7	**-0.02**	**0.29**	**0.89**	2/7	0.19	0.61	1.66
Sensorless Drive Diagnosis	7/7	-0.4	**0.15**	**13.07**	0/7	**-0.43**	3.68	40.41
Sensorless Drive Diagnosis (normalized)	6/7	**0.45**	**1.8**	**4.92**	1/7	0.64	2.39	7.9
Online News Popularity	6/7	**0.57**	**1.78**	**6.42**	1/7	0.95	2.97	22.41
Gas Sensor Array Drift	7/7	**-0.04**	**0.74**	**3.83**	0/7	0.15	3.23	9.6
3D Road Network	2/7	**0.04**	**0.39**	1.7	5/7	0.05	0.42	**1.21**
Skin Segmentation	6/7	**0.21**	**1.93**	**7.02**	1/7	0.25	3.56	10.32
KEGG Metabolic Relation Network (Directed)	5/7	**-0.38**	**0.35**	**3.13**	2/7	-0.37	2.2	32.84
Shuttle Control	8/8	**-0.75**	**0.08**	**6.07**	0/8	0.11	5.59	32.58
Shuttle Control (normalized)	6/8	**1.15**	3.15	**8.1**	2/8	1.16	**2.78**	16.0
EEG Eye State	8/8	**-0.02**	**0.01**	68938.59	0/8	0.54	2.64	**7.02**
EEG Eye State (normalized)	8/8	**-0.06**	**-0.03**	**5.34**	0/8	**-0.06**	8.74	36.58
Pla85900	4/7	0.1	**0.31**	**0.93**	3/7	**0.08**	0.43	1.44
D15112	5/7	**0.1**	**0.25**	**0.75**	2/7	0.11	0.52	4.33
Overall Results	122/165	**0.09**	**0.67**	3000.59	43/165	0.21	2.05	**12.99**

Table 6.3: Baseline convergence times $\bar{t}$ (in seconds) resulting from the comparison of BigVNSClust with Big-means

Dataset	BigVNSClust				Big-means			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	0/7	8.01	19.64	28.35	6/7	**4.22**	**10.68**	**24.38**
HEPMASS	1/7	**2.75**	5.53	14.14	5/7	2.89	**4.17**	**6.36**
US Census Data 1990	0/7	0.18	0.94	2.55	5/7	**0.16**	**0.52**	**2.01**
Gisette	0/7	6.92	7.35	7.98	7/7	**5.18**	**5.47**	**6.64**
Music Analysis	0/7	0.64	3.07	**6.58**	7/7	**0.28**	**2.05**	6.95
Protein Homology	2/7	0.59	**1.22**	**2.5**	5/7	**0.39**	1.25	2.72
MiniBooNE Particle Identification	0/7	**0.68**	1.79	3.44	5/7	0.74	**1.23**	**2.44**
MiniBooNE Particle Identification (normalized)	0/7	0.24	0.48	0.91	7/7	**0.08**	**0.24**	**0.69**
MFCCs for Speech Emotion Recognition	0/7	0.14	0.42	**0.69**	6/7	**0.09**	**0.29**	0.81
ISOLET	0/7	0.4	2.51	4.29	6/7	**0.21**	**1.17**	**2.83**
Sensorless Drive Diagnosis	0/7	0.15	0.55	1.25	6/7	**0.13**	**0.3**	**0.79**
Sensorless Drive Diagnosis (normalized)	0/7	0.06	0.2	0.33	7/7	**0.01**	**0.07**	**0.26**
Online News Popularity	0/7	0.17	0.33	0.61	7/7	**0.07**	**0.22**	**0.53**
Gas Sensor Array Drift	0/7	0.53	0.87	**1.55**	5/7	**0.11**	**0.57**	1.71
3D Road Network	0/7	0.12	0.37	0.63	7/7	**0.06**	**0.21**	**0.48**
Skin Segmentation	0/7	0.06	0.15	0.23	6/7	**0.02**	**0.08**	**0.17**
KEGG Metabolic Relation Network (Directed)	0/7	0.1	0.39	1.0	6/7	**0.06**	**0.24**	**0.88**
Shuttle Control	0/8	0.22	0.44	**0.91**	7/8	**0.04**	**0.32**	1.08
Shuttle Control (normalized)	0/8	0.23	0.26	0.32	8/8	**0.04**	**0.13**	**0.27**
EEG Eye State	1/8	0.08	0.4	**0.78**	7/8	**0.05**	**0.35**	1.05
EEG Eye State (normalized)	0/8	0.02	0.12	**0.57**	5/8	**0.01**	**0.02**	0.72
Pla85900	0/7	0.16	0.66	**1.25**	7/7	**0.03**	**0.39**	1.31
D15112	0/7	0.48	0.8	1.22	7/7	**0.05**	**0.35**	**1.0**
Overall Results	4/165	1.0	2.11	3.57	144/165	**0.65**	**1.32**	**2.87**

6.4.5 Analysis of obtained results

The summarized experimental results reveal that incorporating the VNS metaheuristic into the Big-means scheme leads to a significant boost of the clustering accuracy ε. Concretely, the accuracy increased by more than threefold in comparison to the original Big-means algorithm.

This result aligns with the advanced parallel versions of Big-means (HPClust) discussed in Chapter 4, such as HPClust-competitive and HPClust-cooperative. Although the clustering quality achieved by HPClust-hybrid is slightly superior to that of BigVNSClust, the performance of BigVNSClust is nonetheless remarkable. It demonstrates that even a minor modification in the logic of Big-means can yield significant improvements without the need for advanced parallelization techniques. Furthermore, the integration of a new advanced parallelization method could enhance BigVNSClust's performance even further, potentially surpassing that of HPClust-hybrid. Exploring such a parallelization method may be an exiting direction for future research.

In spite of the fact that BigVNSClust exhibited slightly worse results than Big-means with respect to the baseline convergence time $\bar{t}$, its time results are still much better than those of all other advanced parallel HPClust versions from Chapter 4. One must also acknowledge the fact that the current average time result of 2.11 seconds achieved by BigVNSClust is practically acceptable in most cases. Thus, BigVNSClust can be readily recommended for immediate use by practitioners as a powerful tool for solving industry-level big data clustering problems.

The higher resulting accuracy of BigVNSClust is attributed to the novel shaking properties introduced by the VNS metaheuristic. This iterative shaking of the incumbent solution addresses a key issue with Big-means, discussed in Section 3.7 of Chapter 3. Specifically, Big-means inherently possesses a natural shaking property due to its iterative sampling approach. The addition of VNS further enhances this by iteratively reassigning a portion of the incumbent centroids. This process not only shakes the solution landscape more intensely

but also allows centroids from distinct, non-overlapping clusters to transition between each other's clusters, facilitating better solution exploration.

It is worth noting that the advanced parallel schemes also naturally tackle the aforementioned problem, thus achieving accuracy results comparable to BigVNSClust. Specifically, each advanced parallel scheme employs a unique tactic to increase the likelihood of selecting a centroid configuration where no multiple centroids are placed within a single well-separated cluster. For example, HPClust-competitive accomplishes this by running numerous independent clustering processes with various initializations and ultimately selecting the best result. HPClust-cooperative does so by thoroughly exploring the optimal initial configuration from a vast array. Lastly, HPClust-hybrid effectively combines these two approaches.

6.5 Conclusion and future research

In this chapter, we proposed the BigVNSClust algorithm. This is an enhanced version of the Big-means algorithm based on the VNS metaheuristic. The basics of VNS were reviewed, and the most efficient choice of VNS ingredients has been empirically established. The conducted experimental analysis confirmed that the enhanced algorithm provides a significant increase in the obtained accuracy over the vanilla Big-means approach (with inner parallelism), being on par in this performance metric with other advanced HPClust parallel schemes (competitive, cooperative, and hybrid). Also, BigVNSClust showed a much better time efficiency compared to the advanced parallel HPClust versions, while being only slightly inferior to HPClust-inner in that regard.

These results convince us that the VNS metaheuristic favorably combines with the natural shaking properties of simple Big-means to produce a novel algorithm with highly desirable practical qualities. This result was achieved at the cost of only a tiny increase in the algorithmic complexity. Exploring the advanced way to parallelize BigVNSClust is a promising future research direction that can make the algorithm even more effective and efficient.

More possible future research directions include: the development of a tool that would allow to automatically choose the appropriate parameter triple $(p_{\max},\ s,\ T)$ for each individ-

ual dataset, extension of the algorithm to the clustering paradigms other than MSSC, and exploring what combinations of other well-known metaheuristics can lead to an even more significant improvement of Big-means.

Chapter 7

SUPERIOR PARALLEL BIG DATA CLUSTERING WITH COMPETITIVE STOCHASTIC SAMPLE SIZE OPTIMIZATION IN BIG-MEANS

7.1 Introduction

The purpose of this chapter is to introduce a novel parallel Big-means clustering algorithm with competitive stochastic sample size optimization, BigOptimaS3, and evaluate its performance. The rest of the chapter is structured as follows. Section 7.2 is a brief overview of the main technologies employed in the new algorithm. Section 7.3 presents the proposed method in detail. Section 7.4 describes the experimental setup used to evaluate the performance of the algorithm. Section 7.5 discusses the results and compares the new algorithm with existing methods. Finally, Section 7.6 concludes the chapter and suggests directions for future work.

7.2 Overview of the new algorithm

Competitive optimization is a methodology within optimization problems where multiple solutions or agents compete with each other to achieve the best results [SKT16]. The "competition" among these solutions or agents leads to a dynamic and adaptive process that continually seeks to find better solutions [AL07]. In the context of parallel computing or machine learning, this could mean different parallel processes ("workers") or algorithms competing against each other. Each of these processes or algorithms tries to find a better solution than the others, and through this competition, the overall system pushes towards an optimal or near-optimal solution.

Stochastic sampling is a process of selecting a subset of data from a larger dataset,

where each data point in the larger dataset has an equal probability of being chosen in the sample [HTF09]. This process is "stochastic" or "random" because the selection of data points is based on chance (i.e., the inherent randomness in the system) and not on any specific characteristic of the data. Stochastic sampling is a crucial technique in machine learning and data analysis, especially when dealing with large datasets. It allows for a manageable amount of data to be used for training models or conducting analysis, reducing computational load while still providing a representative subset of the data. In proposed algorithm, stochastic sampling is used to select a subset of data points for each worker to perform the K-means clustering. The size of the subset (i.e., the sample size) is adjusted dynamically based on the algorithm's performance, leading to a more efficient and effective clustering process.

This chapter introduces a new algorithm, BigOptimaS3, which is a parallel version of the Big-means clustering algorithm equipped with competitive stochastic sample size optimization [Mus+23; MM23]. This method innovatively combines principles from parallel computing and stochastic optimization, performing clustering in a parallel manner and adjusting the sample size stochastically in a competitive manner across different workers. In the context of the proposed algorithm for K-means clustering, competitive optimization is a process where different "workers" (or parallel processes) compete against each other. Each worker uses a different sample size to perform the K-means clustering, and they compete to find the sample size that gives the best clustering results. This competition can lead to a more efficient and effective overall clustering solution. Experimental results indicate that this new algorithm is more effective, efficient and scales better with large datasets than traditional K-means and all of its variants.

7.3 Proposed algorithm

7.3.1 Brief description

We propose a novel strategy to parallelize the Big-means clustering algorithm, wherein the size s of the clustered sample from big data used at each iteration varies. The conceptual

Algorithm 27: High-Level BigOptimaS3 (Parallel Big-means Clustering with Competitive Stochastic Sample Size Optimization)

```
Initialize parameters and variables for each worker;
while Not all workers have reached the maximum number of iterations do
    foreach parallel worker w do
        s_w ← Random integer in [s_min, s_max] for each worker w;
        Recalculate value of objective function for incumbent centroids using new s_w;
        Execute Big-means clustering for p iterations using samples of size s_w;
        if improvement in objective function then
            Update incumbent centroids and objective function;
            Add s_w to list L;
        end
    end
end
Analyze the distribution of successful sample sizes in L and select the most effective
  one, s_opt;
Draw a new sample of size s_opt from the dataset;
foreach parallel worker do
    Recalculate the objective function value with s_opt using the new sample;
end
Identify the worker with the best final objective function value and adopt its
  centroids and assignments;
return The best centroids, cluster assignments, and optimal sample size s_opt;
```

Algorithm 28: Detailed BigOptimaS3 (Parallel Big-means Clustering with Competitive Stochastic Sample Size Optimization)

Function `BigOptimaS3`(X, k, s_{min}, s_{max}, T)**:**
  Input : Feature vectors $X = \{x_1, \ldots, x_m\}$;
    Desired number of clusters k;
    Minimal sample size s_{min};
    Maximal sample size s_{max};
    Maximum time T;
  Output: Centroids $C_{\text{best}} = \{c_1, \ldots, c_k\}$;
    Point-to-cluster assignments $Y = \{y_1, \ldots, y_m\}$;
    Optimal sample size s_{opt}.
  $C_w \leftarrow$ Mark all k centroids as degenerate for each worker w;
  $\hat{f}_w \leftarrow \infty$ for each worker w;
  $t_w \leftarrow 0$ for each worker w;
  $L \leftarrow$ Empty list;
  while $t_w < T$ *for any worker* w **do**
    for *each parallel worker* w **do**
      $s_w \leftarrow$ Random integer in $[s_{min}, s_{max}]$;
      Recalculate $\hat{f}_w$ with the new s_w;
      $p_w \leftarrow 0$;
      while $p_w < p$ **do**
        $S_w \leftarrow$ Random sample of size s_w from X;
        for *each* $c \in C_w$ **do**
          if c *is the centroid associated with a degenerate cluster* **then**
            Reinitialize c using K-means++ on S_w;
          end
        end
        $C_{\text{new},w} \leftarrow$ K-means clustering on S_w with initial centroids C_w;
        if $f(C_{new,w}, S_w) < \hat{f}_w$ **then**
          $C_w \leftarrow C_{\text{new},w}$;
          $\hat{f}_w \leftarrow f(C_{\text{new},w}, S_w)$;
          Add s_w to list L;
        end
        $p_w \leftarrow p_w + 1$;
      end
      $t_w \leftarrow t_w + 1$;
    end
  end
  Analyze distribution of s_i values in list L;
  $s_{opt} \leftarrow s_i$ value with highest probability of improving objective function;
  $S \leftarrow$ Random sample of size s_{opt} from X;
  for *each parallel worker* w **do**
    Recalculate $\hat{f}_w$ with s_{opt} using S;
  end
  $C_{\text{best}} \leftarrow$ Centroids of the worker with the smallest $\hat{f}_w$ value;
  $Y \leftarrow$ Assign each point in X to nearest centroid in C_{best};
  return C_{best}, Y, s_{opt};

(high-level) pseudocode for this strategy is shown in Algorithm 27. Within this parallelization strategy, during the algorithm's initialization stage, each of the w workers randomly selects their sample size s_w from the permissible range $[s_{min}; s_{max}]$. Subsequently, each worker operates in parallel for p iterations, adhering to the standard Big-means algorithm's scheme using the allocated sample size s_w.

After p iterations, each i-th worker is assigned a new random sample size s_w. Simultaneously, the value of the target criterion for the worker's current incumbent solution is recalculated to reflect the change in s_w, as any change in s_w necessitates such a recalculation. As the algorithm operates, comprehensive statistics relating to improvements in the objective function's value for a given s_w are collected for all workers. Essentially, each worker, upon witnessing an improvement in the value of its objective function, contributes its current s_w value to a shared list.

By the end of the algorithm's execution, we obtain a list of all s_w values that led to improvements in the objective function's value. By analyzing the distribution of values in this list, we select the s_{opt} value from the list that offers the highest probability of enhancing the objective function's value. The desired result can be achieved by calculating a simple mean of the given list, which corresponds to the expected value of the improving sample size.

This strategy effectively creates a competitive environment among workers, allowing for simultaneous variation in the used sample size s_w and subsequent determination of its optimal value.

7.3.2 Detailed algorithm description

The algorithm presented in Algorithm 28 details a parallel implementation of the K-means clustering method with competitive stochastic sample size optimization — BigOptimaS3. The method aims to determine the best cluster centroids and assign data points to these clusters efficiently.

The algorithm initializes by marking all k centroids as degenerate for each worker w. Each worker also has an initial best-so-far objective function value $\hat{f}_w$ and iteration count t_w

set to ∞ and 0, respectively. An empty list L is also defined to keep track of sample sizes that lead to improvement in the objective function.

In each iteration of the main loop, every worker operates in parallel, choosing a random sample size s_w between s_{min} and s_{max}, and recomputing the $\hat{f}_w$ with the new sample size. Then, within the defined maximum number of passes p, the worker takes a random sample of size s_w from the data set X and processes the centroids. For each centroid c associated with a degenerate cluster, the worker reinitializes c using K-means++ on the sample S_w. The worker performs K-means clustering on S_w with initial centroids C_w to get new centroids $C_{\text{new},w}$. If the new centroids result in a better objective function value, the worker updates its current centroids, best-so-far objective function value, and adds the sample size to list L. The pass counter p_w is then incremented.

After all workers finish their iterations or reach the maximum number of iterations T, the algorithm proceeds to analyze the distribution of sample sizes in list L. The algorithm selects the sample size s_{opt}, which has the highest probability of improving the objective function. This selection can be achieved by calculating the simple mean of the values in the list L. Then, a new sample of size s_{opt} is taken from the dataset X, and each worker recalculates its best-so-far objective function value with this new sample size.

Finally, the algorithm chooses the centroids C_{best} of the worker with the smallest $\hat{f}_w$ value, assigns each point in X to its nearest centroid in C_{best}, and returns these centroids, the cluster assignments, and the optimal sample size s_{opt}.

In this work, we assume that each worker has equal access to the full-sized dataset and can independently draw samples from it. For the sake of simplicity, in this study we are not exploring various available opportunities for further optimization of the algorithm, particularly those concerning distributed data storage across different nodes of the computing system. Such optimizations merit a separate study.

7.3.3 Algorithm properties

The proposed parallel Big-means clustering algorithm with competitive stochastic sample size optimization algorithm is designed with several key properties to enhance the performance of traditional K-means clustering, as demonstrated in Algorithm 28.

The *parallelization* property allows multiple workers to perform K-means clustering simultaneously on different subsets of the data. This feature significantly enhances the algorithm's computational efficiency and enables it to handle large datasets effectively.

Sample size optimization is integrated into the algorithm's design, with each worker randomly selecting a different sample size for each iteration. This stochastic sampling brings diversity and adaptability into the clustering process, enhancing its ability to find the global optimal solution.

To mitigate the issue of *degenerate clusters*, which could hinder the algorithm's effectiveness, our approach periodically re-initializes the centroids of such clusters using K-means++. This action ensures that all clusters remain relevant and contribute to finding the optimal solution.

Our algorithm also incorporates a *competitive strategy* among parallel workers, each utilizing different sample sizes. Through competition, the workers stimulate increased efficiency and optimization within the overall algorithm.

The *global best solution* is maintained by continually comparing and updating the best clustering result among all workers. This practice ensures the algorithm converges towards an optimal or near-optimal solution.

Scalability is a core feature of our algorithm, achieved by processing smaller subsets of the data in each iteration. This approach significantly reduces the computational demands and makes the algorithm suitable for large-scale applications.

Regarding *efficiency*, the algorithm's iterative process and concurrent execution of workers allow it to rapidly converge towards the optimal solution. In addition, the competitive nature of the algorithm encourages each worker to seek out the most effective sample size,

further enhancing its efficiency.

The proposed algorithm is *robust*, capable of handling data of various types and sizes. Its design elements, such as stochastic sampling and dynamic sample size adjustments, enable it to adapt to diverse data characteristics.

Lastly, the *natural data-driven shaking of the incumbent solution* is an inherent feature of the proposed algorithm. By creating a new sample in each iteration, it introduces variability into the clustering results and prevents the algorithm from being trapped in suboptimal solutions.

7.4 Experimental evaluation

For the information on the used datasets, software and hardware, experimental methodology, evaluation metrics, as well as other implementation details, we refer the reader to Section 4.6 of Chapter 4. In this chapter, the design of the experiments is identical to Chapter 4, except for some minor differences that we describe below.

7.4.1 Range of algorithms

We compare the proposed algorithm with the hybrid-parallel version of the Big-means algorithm [MM23] (HPClust-hybrid), which resulted with the best performance among all the K-means-like algorithms so far.

7.4.2 Time metric

For both algorithms, the average CPU time, denoted as $\bar{t}$, corresponds to the time required for convergence to the baseline results, as outlined in Section 4.6. The rationale for preferring this metric over the straightforward total clustering time is detailed in Section 4.6. Moreover, using the same time metric $\bar{t}$ as in Chapters 4 and 6 ensures an additional cross-comparison with the results presented in those chapters. The values of $\bar{t}$ are expressed in seconds.

7.4.3 Other implementation details

For a fair comparison, the hybrid version of Big-means was configured with the same values of parameters s and T that were used in the experiments of Chapter 3 and 4.

For the proposed BigOptimaS3 algorithm, s_{min} was chosen to be 0.5 of the sample size s used in Big-means, while s_{max} was chosen to be 2 times s (or m, if this number exceeds m for the given dataset). For each pair (X, k), the choices of parameters s, T and n_{exec} precisely matched the values specified in the original Chapter 3 and the Big-means paper [Mus+23].

7.4.4 Experimental results

The total number of conducted experiments reached 7,366. Tables 7.1 – 7.2 display the summarized results.

7.5 Experimental results and discussion

7.5.1 Performance evaluation

A summary of the results of the main experiment are provided in Tables 7.1 – 7.2. Based on the experimental results, it was observed that the proposed algorithm performed consistently better than Big-means on all datasets, both with respect to the accuracy and time.

We attribute the outstanding performance of the proposed algorithm to its ability to approximate the probability distribution of sample sizes that improve the sample objective function. In the competitive parallelization strategy, each worker starts with its own K-means++ initialization, which strengthens the final result via diversification. Also, due to this kind of parallelism, the algorithm is able to accumulate the necessary statistics for the approximation in a very timely efficient manner. Then, the optimal sample size value s_{opt} can be obtained by evaluating the simple mean of the accumulated sample size occurrence distribution.

The value s_{opt} is the size of a sample that attains the best balance between sparsifying high-density clusters while still including enough mass of low-density clusters into the sam-

Table 7.1: Relative clustering accuracies ϵ (in %) resulting from the comparison of BigOptimaS3 with Big-means

Dataset	BigOptimaS3				Big-means			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	7/7	**-0.01**	**0.04**	**0.08**	0/7	0.03	0.08	0.13
HEPMASS	7/7	**0.0**	**0.06**	**0.14**	0/7	0.03	0.09	0.24
US Census Data 1990	6/7	**0.32**	**1.25**	**2.21**	1/7	0.39	1.38	2.81
Gisette	7/7	**-0.52**	**-0.47**	**-0.42**	0/7	-0.44	-0.39	-0.31
Music Analysis	7/7	**0.15**	**0.32**	**0.55**	0/7	0.46	0.82	2.01
Protein Homology	7/7	**0.01**	**0.12**	**0.52**	0/7	0.4	0.88	2.18
MiniBooNE Particle Identification	7/7	**-0.09**	**-0.05**	0.45	0/7	-0.05	0.0	**0.39**
MiniBooNE Particle Identification (normalized)	7/7	**0.06**	**0.15**	**0.31**	0/7	0.26	0.66	1.55
MFCCs for Speech Emotion Recognition	7/7	**0.03**	**0.13**	**0.7**	0/7	0.13	0.37	0.92
ISOLET	7/7	**-0.11**	**0.08**	**0.31**	0/7	0.04	0.24	0.87
Sensorless Drive Diagnosis	7/7	**-0.43**	**-0.23**	**1.16**	0/7	-0.38	-0.19	32.12
Sensorless Drive Diagnosis (normalized)	7/7	**0.21**	**0.9**	**2.16**	0/7	0.39	1.81	4.71
Online News Popularity	7/7	**0.13**	**0.74**	**1.95**	0/7	0.62	1.71	4.97
Gas Sensor Array Drift	7/7	**-0.1**	**0.53**	**1.88**	0/7	-0.01	0.79	4.02
3D Road Network	7/7	**0.01**	**0.05**	**0.18**	0/7	0.05	0.25	1.08
Skin Segmentation	7/7	**-0.23**	**0.65**	**2.69**	0/7	0.01	1.63	5.47
KEGG Metabolic Relation Network (Directed)	7/7	**-0.4**	**0.17**	**0.92**	0/7	-0.32	0.32	6.9
Shuttle Control	8/8	**-0.52**	**0.52**	**2.03**	0/8	-0.14	1.52	7.15
Shuttle Control (normalized)	8/8	**0.49**	**0.87**	**2.38**	0/8	0.83	1.99	5.14
EEG Eye State	8/8	**-0.01**	**0.55**	**0.66**	0/8	0.52	0.58	6.13
EEG Eye State (normalized)	7/8	**-0.06**	**-0.02**	**0.19**	1/8	-0.05	0.0	80.04
Pla85900	7/7	**0.01**	**0.05**	**0.41**	0/7	0.07	0.2	0.73
D15112	7/7	**-0.0**	**0.04**	**0.19**	0/7	0.08	0.18	0.51
Overall Results	163/165	**-0.05**	**0.28**	**0.94**	2/165	0.13	0.65	7.38

Table 7.2: Baseline convergence times $\bar{t}$ (in seconds) resulting from the comparison of BigOptimaS3 with Big-means

Dataset	BigOptimaS3				Big-means			
	#Succ	Min	Median	Max	#Succ	Min	Median	Max
CORD-19 Embeddings	3/7	13.38	**15.1**	**17.42**	4/7	**12.6**	17.25	24.16
HEPMASS	2/7	4.35	4.91	8.78	5/7	**2.24**	**4.23**	**8.01**
US Census Data 1990	2/7	1.9	2.3	3.18	5/7	**0.31**	**0.7**	**1.51**
Gisette	7/7	**13.41**	**16.17**	**19.79**	0/7	24.03	25.98	31.79
Music Analysis	6/7	**2.22**	**3.51**	6.88	1/7	2.76	3.97	**6.84**
Protein Homology	5/7	**2.1**	**3.72**	5.53	2/7	3.03	3.83	**5.42**
MiniBooNE Particle Identification	3/7	5.33	6.01	7.58	4/7	**5.18**	**5.96**	**7.02**
MiniBooNE Particle Identification (normalized)	6/7	**0.43**	**0.7**	**1.04**	1/7	0.47	0.77	1.25
MFCCs for Speech Emotion Recognition	6/7	**0.31**	**0.69**	**1.15**	1/7	0.61	0.78	1.21
ISOLET	7/7	**1.38**	**2.17**	**3.29**	0/7	2.24	2.88	3.99
Sensorless Drive Diagnosis	3/7	1.54	2.0	2.62	4/7	**1.41**	**1.97**	**2.45**
Sensorless Drive Diagnosis (normalized)	7/7	**0.05**	**0.19**	0.48	0/7	0.09	0.29	**0.4**
Online News Popularity	5/7	**0.25**	**0.48**	**0.82**	2/7	0.42	0.59	0.98
Gas Sensor Array Drift	7/7	**0.45**	**0.77**	**1.27**	0/7	0.76	1.18	2.18
3D Road Network	6/7	**0.32**	**0.68**	**1.15**	1/7	0.55	0.81	1.25
Skin Segmentation	7/7	**0.02**	**0.07**	**0.29**	0/7	**0.02**	0.18	0.32
KEGG Metabolic Relation Network (Directed)	3/7	0.63	0.86	1.19	4/7	**0.59**	**0.83**	**1.12**
Shuttle Control	4/8	**0.3**	**0.61**	1.1	4/8	0.35	**0.61**	**0.92**
Shuttle Control (normalized)	8/8	**0.01**	**0.06**	**0.24**	0/8	0.02	0.27	0.39
EEG Eye State	7/8	**0.18**	**0.48**	1.07	1/8	0.26	0.56	**0.96**
EEG Eye State (normalized)	2/8	0.17	0.27	0.41	6/8	**0.14**	**0.25**	**0.39**
Pla85900	7/7	**0.03**	**0.17**	**0.45**	0/7	0.21	0.47	1.12
D15112	7/7	**0.02**	**0.08**	**0.25**	0/7	0.12	0.39	0.77
Overall Results	120/165	**2.12**	**2.7**	**3.74**	45/165	2.54	3.25	4.54

ple. In addition to the accumulation of the improving sample size distribution, competitive workers are able to dynamically guide the flow of centroids through unfavorable situations by using various random sample sizes in the range $[s_{min}, s_{max}]$. For instance, these unfavorable situations might occur when a sample has large gaps between highly dense clusters (thus preventing the fluidity of centroids between them) or excludes some clusters due to an overly intense sparsification.

The median values of parameter s_{opt}, relative to the number of clusters, were obtained for the datasets as follows: 39610, 81648, 7481, 9497, 7442, 70865, 130001, 15050, 14965, 4910, 58501, 4378, 12418, 9261, 125238, 9956, 53357, 57951, 2482, 14979, 14979, 17502, 9559. The enumeration order reflects dataset sizes from largest to smallest, as found in [Mus+23] (Chapter 3) and [MM23] (Chapter 4), where the original s values are available for comparison.

7.6 Conclusion and future research

In this chapter, we proposed BigOptimaS3, a parallel Big-means clustering algorithm with competitive stochastic sample size optimization, as well as thoroughly evaluated its performance against the state-of-the-art hybrid Big-means algorithm (HPClust-hybrid) using a wide array of real-world datasets. The idea of using an automatic procedure for approximating the optimal sample size stemmed from multiple considerations. First, it is practically hard and error-prone to estimate the sample size for real-world big datasets. Second, using a fixed sample size makes the iterative improvement nature of the Big-means algorithm too rigid. Indeed, sampling with a fixed sample size is limited in the flexibility of approximating and sparsifying regions of the dataset with different densities.

Our improved version of Big-means exhibited exceptional results both in the resulting quality and time, pushing much further the state of the art in the field of big data clustering. We are confident that our work presents a valuable contribution to the scientific field, as well as brings a tool of considerable practical value to practitioners in the field of big data.

For future research, we plan to investigate other dimensions for experimentation, including different ways to explore the range $[s_{min}, s_{max}]$ and exploit the currently best obtained

sample size across iterations.

Chapter 8

VARIABLE LANDSCAPE SEARCH

8.1 Motivation and chapter structure

In the preceding chapters, we have identified a set of recurring concepts and meta-ideas, which we now synthesize into the novel Variable Landscape Search (VLS) metaheuristic. This chapter's contribution lies in the articulation and analysis of this innovative optimization metaheuristic, which balances generality with practical specificity. In Section 8.8, we demonstrate how the findings from earlier sections of this dissertation are effectively integrated within the VLS framework. Furthermore, we establish that existing powerful metaheuristics, such as Variable Formulation Search (VFS) [Par+13], represent specific instances within the broader VLS paradigm.

The structure of this chapter is as follows. Section 8.2 introduces essential concepts, further elucidating the motivation behind the proposed VLS approach. Section 8.3 provides a comprehensive review of existing metaheuristic methodologies in the literature, highlighting the distinctiveness of our proposed VLS metaheuristic. The concepts of landscape meta-space and landscape neighborhood are thoroughly explored in Sections 8.4 and 8.5, respectively. Section 8.6 delves into the core ideas and procedures of the Basic Variable Landscape Search (BVLS) method, which is detailed in Section 8.7. Also, Section 8.7 emphasizes the unique advantages and features of VLS. Section 8.8 illustrates how both the results of this dissertation and other metaheuristic strategies can be framed as specific applications within the versatile VLS framework. Finally, Section 8.9 concludes the chapter, offering insights into potential avenues for future research.

8.2 Introduction

Metaheuristic methods are higher-level heuristics designed to select and modify simpler heuristics, with the goal of efficiently exploring a solution space and finding near-optimal or optimal solutions [SSG18]. They incorporate strategies to balance local search (exploitation of the current area in the solution space) and global search (exploration of the entire solution space), in order to escape from local optima and reach the global optimum [CDC21]. Incorporating a mechanism to adaptively change the search space, the proposed Variable Landscape Search (VLS) approach can be classified as a metaheuristic method, given that it embodies these core elements of metaheuristic optimization strategies.

Search space is the space of all potential solutions for a problem. Since the goal of a typical optimization problem is to find a global optimum, and any global optimum is necessarily a local optimum, we can restrict our notion of search space to include only all the local optima for the current objective function landscape [LS16]. In fact, any feasible solution can be mapped to the nearest locally optimal solution via the appropriate local search [VS23].

In the context of an optimization problem, there are two primary ways to influence the search space:

1. *Modifying the input data:* The input data are the raw details, observations, or parameters on which the problem is based. Altering these inputs can change the landscape of the objective function, thereby the distribution of optimal solutions as well [AH17]. For example, in a resource allocation problem, changing the quantity of available resources would directly impact the resulting optimal solutions [DGK17];

2. *Altering the problem formulation:* This involves changing the way the problem is structured or defined [MBU22]. The formulation of a problem includes its objectives, constraints, and the relationships between different variables. Tweaking any of these aspects can lead to a different set of solutions that are considered viable.

These two elements – the input data and the problem formulation – jointly determine

the objective function landscape and the corresponding search space. To visualize this relationship, imagine a process where the combination of specific input data and a particular problem formulation leads to a distinct set of possible locally optimal solutions.

In more detail, the input data is a subset of a larger, overarching data space. This overarching space contains all possible data points that could be considered for problems of this type. The specific data used for a particular problem is a selection from this vast pool.

Similarly, the problem formulation is chosen from a set of all possible ways to frame or define the problem. This set includes every conceivable method to state the objectives, constraints, and relationships for problems of a similar nature.

The search space, in turn, can be understood as the set of all locally optimal solutions that satisfy the constraints and objectives as defined by the chosen problem formulation, using the selected input data. This space is critical in optimization as it contains every solution that could potentially be the "best" or "optimal" solution, depending on the problem's objectives.

A *neighborhood solution* is a solution lying within the defined "neighborhood" of a given current solution in the solution space. The concept of a "neighborhood" typically relates to a defined proximity or closeness to a given solution [Alt+14]. The determination of what constitutes a "neighborhood solution" depends on the structure of the problem and the optimization algorithm being used. The process of iteratively exploring these "neighborhood solutions" and moving towards better solutions forms the core of local search algorithms and many metaheuristic methods [LMS19]. The aim is to gradually improve upon the current solution by exploring its immediate vicinity in the solution space.

A *neighborhood structure* refers to a method or function that can form a set of potential solutions that are adjacent or "near" to a given solution in the solution space. The concept of "nearness" or adjacency is defined based on the problem at hand and the specific algorithm being used [Alt+14]. For instance, in a combinatorial optimization problem, two solutions might be considered neighbors if one can be reached from the other by a single elementary operation, such as swapping two elements in a permutation.

Neighborhood structure is a fundamental concept in many local search and metaheuris-

tic algorithms, such as variable neighborhood search (VNS) [Han+17], simulated annealing [SA16], tabu search [GP19], and genetic algorithms [Kra17]. These methods rely on iteratively exploring the neighborhood of the current solution to try to find a better solution. The definition of the neighborhood structure directly impacts the effectiveness of the search process, the diversity of solutions explored, and the ability of the algorithm to escape local optima. For example, in VNS, the order and rule of altering neighborhoods throughout the search process are significant factors to build the efficient heuristic [Dua+18].

Assume that for some choice of input data and problem formulation, the optimization task became to minimize objective function f over feasible solution space (feasible region) S. Then, the corresponding *fixed objective function landscape* is the fixed set

$$L_f^S = \{(x, f(x)) \mid x \in S\}$$

of all feasible solutions paired with the corresponding objective function values that remains constant throughout the optimization process. This set coincides with the graph of the objective function when the function's domain is restricted to the feasible region.

From now on, the terms "landscape" or "objective landscape" will simply refer to a fixed objective function landscape. Also, we will use the simplified notation L to refer to an arbitrary landscape when its underlying objective function and feasible space are not important for the context.

For a given landscape L_f^S, its *search space* S^* is defined to be the subset

$$S^*\left(L_f^S\right) = \{x \in S \mid x \text{ is a local optimum}\} \subseteq S$$

of all the local optima of the objective function contained in the feasible space. This static search space is characterized by its unchanging nature, where every possible locally optimal solution is defined at the outset of the optimization procedure. All solutions in this space are accessible to be tested, evaluated, or used at any point during the problem-solving process, and no new solutions are introduced or existing ones excluded once the optimization process

has commenced.

In the context of optimization algorithms, a *variable objective function landscape* is a landscape whose distribution of objective function values over the feasible region, as well as the feasible region itself, can vary throughout the optimization process. As a result, the landscape's search space is changing as well. Unlike a fixed landscape, where all locally optimal solutions are predefined and static, a variable landscape allows for a degree of modification in the set of potential solutions. Additionally, it can enable the introduction of new potential solutions, or the exclusion of existing ones, based on certain conditions or criteria during the optimization iterations.

The manipulation of a landscape could be based on a variety of strategies, including but not limited to, stochastic sampling, adaptive adjustments, or guided exploration techniques. The primary goal of this dynamism is to enhance the search process, promote diversity, prevent premature convergence to local optima, and increase the likelihood of achieving the global optimum.

The concept of a variable objective function landscape is at the core of the Variable Landscape Search (VLS) metaheuristic, where the algorithm systematically modulates the landscape by utilizing some special approach at each iteration, aiming to bypass local minima and aspire towards the global optimum.

Figure 8.1 provides an illustrative example of how VLS works. The directionality of the iterative search, as represented by arrows in Figure 8.1, is critical. Each arrow signifies an iteration where a new perturbed landscape $f_i(\cdot)$ has been taken, and the algorithm has moved from one solution to another, possibly better one. This creates a trajectory, allowing the visualization of how the algorithm is navigating the adaptive landscape, and how it is making progress towards the global optimum.

8.3 Related work

The concept of manipulating the landscape itself is less common in metaheuristics, but some methods can be interpreted as doing so, either directly or indirectly.

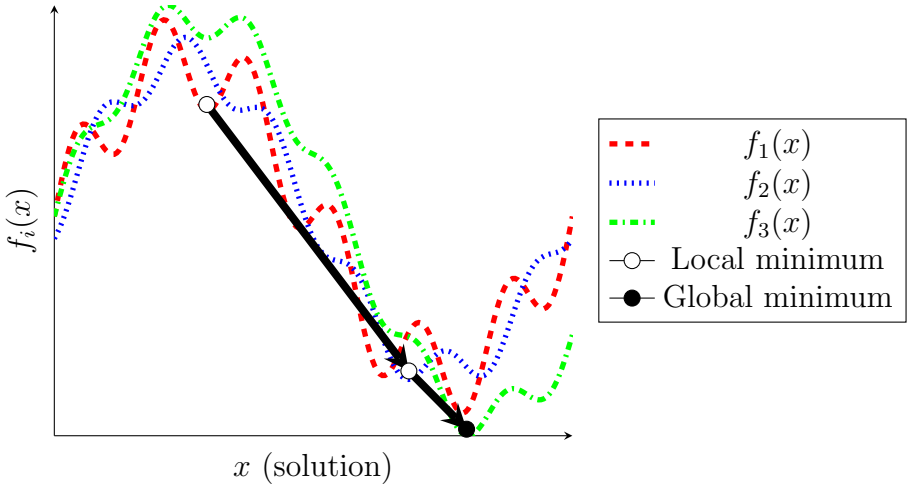


Figure 8.1: Illustration of the process of landscape variation to overcome local minima

1. *Hyper-heuristics [Dra+20]:* These are heuristics to choose heuristics, which could be seen as a form of manipulating the search space of the landscape. Instead of operating on the solutions directly, they operate on a space of heuristics that generate or improve solutions. The choice of a heuristic can significantly change the set of potential solutions (local optima) that are explored;

2. *Constructive metaheuristics*, such as *Ant Colony Optimization (ACO)* [DS19] and *Greedy Randomized Adaptive Search Procedure (GRASP)* [RR19], can be interpreted as manipulating the search space of the landscape. They construct solutions step-by-step, and the choices made in the early steps change the part of the search space that is explored in later steps;

3. *Cooperative co-evolutionary algorithms (CCEAs) [Ma+19]:* In these methods, the solution space is divided into several sub-spaces, and different search processes or different populations evolve in these different sub-spaces. The results are then combined, which can be seen as manipulating the search space;

4. *Variable Neighborhood Search (VNS) [Bri+23]:* In VNS, the neighborhood structure

itself (i.e., the definition of which solutions are "neighbors" and can be reached from a given solution) is varied during the search. This can be interpreted as a manipulation of the search space;

5. *Genetic Algorithms (GA)* with adaptive representation [PM17]: Some versions of GAs use adaptive representation, where the encoding of the solutions (i.e., how solutions are represented as chromosomes) is changed during the search based on some criteria. This can be seen as a manipulation of the search space;

6. *Variable Formulation Search (VFS)* [Par+13] is a dynamic, adaptive representation of problem formulations rather than a static, predefined one. It refers to the flexibility in the formulation of an optimization problem where the structure of the mathematical representation itself can be altered over the course of the problem-solving process. Unlike a static formulation space, where the mathematical model, decision variables, constraints, and objective function are fixed from the start, a variable formulation space allows for adaptive changes in the problem's formulation. This can involve the introduction of new decision variables, the removal or modification of constraints, or even changes in the objective function, based on certain criteria, algorithmic processes, or evolving insights into the problem during the course of the optimization iterations.

In the context of optimization problems, problem formulation typically refers to the mathematical representation of a problem, including decision variables, constraints, and objective function(s). It is this formulation that guides the computational methods applied to solve the problem.

Raw data that will be processed, such as data to be clustered in a clustering problem, is usually considered as an input to the problem rather than a part of the problem formulation itself. However, the specifics of how data is to be processed, including how it is to be represented and handled within the problem, would typically be part of the problem formulation.

For example, in a clustering problem, the problem formulation might specify that a certain distance metric (like Euclidean distance or Manhattan distance) will be used, that a certain number of clusters are to be formed, or that certain constraints must be satisfied by the clusters [Cor13]. The specific data points to be clustered, however, would be an input to the problem as defined by this formulation, not part of the formulation itself.

Variable Landscape Search represents a more encompassing strategy in optimization, broadening the scope beyond what is offered by Variable Formulation Search. While the latter focuses on exploring different objective landscapes by varying the problem's formulation, Variable Landscape Search extends this concept by not only considering changes in problem formulation but also incorporating modifications in the input data. This dual approach allows for a more comprehensive exploration of potential solutions, making Variable Landscape Search a generalization of Variable Formulation Search. It acknowledges that altering the input data is as crucial as changing the formulation in discovering diverse and precise objective landscapes and their corresponding search spaces. We explore the details of how VLS becomes a generalization of VFS and its various versions in Section 8.8. This section contains concrete examples of applying VLS to construct optimization heuristics.

The proposed approach of Variable Landscape Search (VLS), in which the objective function landscape itself is explicitly and systematically varied during the search, represents a novel perspective in the field of metaheuristics.

8.4 Landscape meta-space

The concept of *landscape meta-space* encompasses the aggregation of all potential objective function landscapes pertinent to a specific problem or a set of problems. Within this framework, each individual objective landscape comprises a distinct search space — the set of potential solutions in form local optima for the landscape. These potential solutions correspond to a unique problem formulation, representation, constraint set, as well as specific dataset being processed.

The concept of a landscape meta-space is especially relevant in complex optimization

scenarios, where the problem at hand may be addressed from different perspectives or under different assumptions, each leading to a different landscape of the objective function. As such, the landscape meta-space represents the entire universe of potential solutions across all these diverse contexts.

Thus, landscape meta-space provides a framework for broader exploration and navigation across multiple landscapes, facilitating a more comprehensive search process. It allows for the shifting or manipulation of the search space itself, rather than only exploring within a fixed search space, enhancing the potential to discover superior or more diverse solutions.

The landscape meta-space concept takes the exploration process to a higher level of abstraction, allowing search algorithms to consider not only the local optima within a given objective landscape but also the structure and characteristics of the landscape itself. This opens up new possibilities for innovative search strategies, such as the Variable Landscape Search (VLS) method.

8.5 Landscape neighborhood

A *neighborhood* of a landscape, in essence, encapsulates a subset of the landscape meta-space. This subset is defined by a "radius of similarity" or "proximity parameter" around the current objective landscape. This "neighborhood" represents a region within the landscape meta-space where each individual landscape shares a certain degree of similarity, correlation, or alignment with the current landscape. These connections may be drawn based on a range of factors, such as solution quality, complexity of problem formulation, or even aspects of data characteristics.

This concept provides a local scope within the more global landscape meta-space, enabling a more focused and computationally efficient search. The dynamism inherent to a landscape neighborhood is beneficial as it allows for the exploration of landscapes that are "close" to the current landscape, potentially leading to improvements in solution quality while still maintaining computational feasibility. Through iterative redefinition of the "neighborhood", the algorithm can progressively navigate the landscape meta-space, yielding a trajectory of

landscapes that encapsulates the overall optimization process.

8.6 Variable Landscape Search (VLS) method

For a given optimization problem P, there exists the corresponding space of all its formulations $\mathbb{F}$, which is called the formulation space. Each formulation $F = (f, C) \in \mathbb{F}$ consists of a mathematically defined objective function f, which is to be minimized, as well as a mathematically defined set of constraints C. Alternatively, each set of constraints C can be included into the corresponding objective function f as penalizing terms.

Also, we assume that there exists overarching data space $\mathbb{X}$, which serves as the source of input data for problem formulations. Data space $\mathbb{X}$ can be either finite or infinite, representing a data stream in the latter case.

We have two ways of influencing the objective function landscape L:

1. By changing the input data $X \subseteq \mathbb{X}$;

2. By changing the problem formulation (or model) $F \in \mathbb{F}$.

Thus, we have the landscape evaluation map

$$\mathscr{L} : (X, (f, C)) \mapsto L_f^S,$$

where $X \subseteq \mathbb{X}$ is an input dataset, $F = (f, C) \in \mathbb{F}$ is a formulation for the given problem P, and S is the feasible region resulting from evaluating constraints C on input data X.

The general form of optimization problem that we are looking at may be given as follows:

$$\min \left\{ \; f(x) \mid x \in S, \; L_f^S \in \mathbb{L} \; \right\} \tag{8.1}$$

where $\mathbb{L}$, S, x and f denote, respectively, the landscape meta-space (the space of landscapes), a feasible solution space, a feasible solution, and a real-valued objective function. We restrict the landscape meta-space to the image of the landscape evaluation map $\mathscr{L}$:

$$\mathbb{L} = \mathscr{L}(2^{\mathbb{X}} \times \mathbb{F})$$

Given that duplicates may exist within the landscape meta-space $\mathbb{L}$, we ensure injectivity by assuming that each landscape $L \in \mathbb{L}$ implicitly contains information from its originating input dataset and formulation. More precisely, for every $L \in \mathbb{L}$ there exists the pair $(X, F) = \mathscr{L}^{-1}(L)$.

VLS ingredients

The following three steps are repeated in sequence in each iteration of a VLS-based heuristic until a stopping condition, such as a limit on execution time, is reached:

(1) Landscape shaking procedure;

(2) Improvement procedure;

(3) Neighborhood change.

An initial objective landscape $L_f^S \in \mathbb{L}$ and a feasible solution $x \in S$ are required to start the process, and these starting points are typically selected at random from the corresponding supersets.

Let us outline the steps listed above with respect to the basic VLS (BVLS) framework.

Shaking procedure

In this work, we distinguish between different types of a neighborhood structure: on the landscape meta-space, which is the product of neighborhood structures on the data and formulation spaces, and on a feasible solution space.

For an abstract space E, let $\mathcal{N} = \{\mathcal{N}_{k_{\min}}, \ldots, \mathcal{N}_{k_{\max}}\}$ be a set of operators such that each operator $\mathcal{N}_k$, $k_{\min} \leq k \leq k_{\max}$, maps a given choice of element $e \in E$ to a predefined neighborhood structure on E:

$$\mathcal{N}_k(e) = \{e' \in E \mid \phi(e, e') \leq \Phi_k\},\ k = k_{\min}, \ldots, k_{\max}, \tag{8.2}$$

where $\phi(\cdot,\cdot)$ is some distance function defined on E, and $\Phi = \{\Phi_1, \ldots, \Phi_k\}$ are some positive numbers (integer for combinatorial optimization problems).

Note that the order of operators in $\mathcal{N}$ also defines the order of examining the various neighborhood structures of a given choice of element e, as specified in (8.2). Furthermore, the neighborhoods are purposely arranged in increasing distance from the incumbent element e; that is,

$$\Phi_{k_{\min}} < \Phi_{k_{\min}+1} < \ldots < \Phi_{k_{\max}} \tag{8.3}$$

Alternatively, each operator $\mathcal{N}_k$ may be constant with respect to element e, yielding a neighborhood structure that is independent of the choice of e. Also, the neighborhood structures may well be defined using more than one distance function or without any distance functions at all (e.g., by assigning fixed neighborhoods or according to some other rules).

Now, let us assume that a set of operators $\mathcal{N}^1$ is defined on the data space $\mathbb{X}$, while another set of operators $\mathcal{N}^2$ is defined on the formulation space $\mathbb{F}$. Their parameters can be conveniently accessed from the following matrix:

$$K = \{K_j^i\}_{i\in\{1,2\},j\in\{min,max\}}$$

Then, the neighborhood structure on the landscape meta-space $\mathbb{L}$ can be defined as the Cartesian product:

$$\mathcal{L} = \mathcal{N}^1 \times \mathcal{N}^2$$

A simple shaking procedure consists in selecting a random element either from $\mathcal{N}_k^1(X)$ or $\mathcal{N}_k^2(F)$ depending on the current shaking phase i (see Algorithm 29).

Improvement step

We assume that a constant predefined neighborhood structure N_S exists on each feasible solution space S.

Within an objective function landscape, an improving search typically involves an examination of alternate solutions that are "near" to the current solution x. Thus, the improving

Algorithm 29: Shaking Procedure

Function `Shake_landscape`(L, $\mathcal{L}$, k, i)**:**
  $X, F \leftarrow \mathscr{L}^{-1}(L)$; `// obtain data and formulation of incumbent landscape`
  if $i = 1$ **then**
    $X' \leftarrow$ Choose $X' \in \mathcal{N}_k^1(X)$ at random; `// shake data`
    $F' \leftarrow F$;
  else
    $X' \leftarrow X$;
    $F' \leftarrow$ Choose $F' \in \mathcal{N}_k^2(F)$ at random; `// shake formulation`
  end
  $L' \leftarrow \mathscr{L}(X', F')$; `// evaluate new shaken landscape`
  return L';

search uses local information obtained from a single neighborhood $N_S(x)$ consisting of surrounding solutions, where S is the feasible solution space of the landscape.

This is precisely what the improvement step does in BVLS. Local search typically utilizes either the "best improvement" strategy, which fully explores $N_S(x)$ and updates x with the optimal solution in $N_S(x)$ if it surpasses the current x, or the "first improvement" strategy, which updates x with the first solution in $N_S(x)$ that improves upon x. The process concludes when no better solution than the current x is discovered in $N_S(x)$.

The pseudocode for the best improvement local search procedure is presented in Algorithm 30.

Algorithm 30: Local Search Using Best Improvement

Function `Best_improvement_local_search`(x, L_f^S)**:**
  repeat
    $x' \leftarrow x$; `// remember old solution`
    $x \leftarrow \arg\min_{y \in N_S(x')} f(y)$; `// find best improving solution`
  until $f(x') \leq f(x)$;
  return x';

Neighborhood change step

An iteration of VLS starts with the shake operation, which moves the incumbent landscape L to a perturbed landscape L'. Then, the improvement step moves x to a local minimum x'; that is,

$$f(x') \leq \min\{f(y) : y \in N_S(x')\}. \tag{8.4}$$

At this point, the neighborhood change step takes over to decide how the search will continue in the next iteration of VLS. The routine commonly used is known as the *sequential neighborhood change step*. If $f'(x') < f'(x)$ (the new solution is better than the incumbent with respect to the objective function of the perturbed landscape), then $x \leftarrow x'$, $L \leftarrow L'$ (the new solution with its landscape becomes the incumbent), and $k \leftarrow k_{\min}$ (the shake parameter is reset to its initial value); otherwise $f'(x') \geq f'(x)$ holds, $k \leftarrow k+1$ (increment to the next larger neighborhood or return to the initial one, if k exceeds $k_{\max}$), and $n \leftarrow n+1$ (increment the counter of unsuccessful iterations). The pseudocode for the sequential neighborhood change step is shown in Algorithm 31. Other forms of the neighborhood change step can be also used. These include cyclic, pipe, random and skewed forms [Han+17].

Algorithm 31: Sequential Neighborhood Change Step

```
1  Function Neighborhood_change_sequential(x, x', L = L_f^S, L' = L_f'^S', k, k_min,
     k_max):
2      if f'(x') < f'(x) then
3          x ← x'; // move in solution space
4          L ← L'; // move in landscape space
5          k ← k_min; // reset shaking power
6      else
7          k ← k + 1; // increase shaking power
8          if k > k_max then
9              k ← k_min;
10         end
11     end
```

8.7 Basic Variable Landscape Search

The basic VLS scheme requires some stopping criterion for the main loop. Usually, this can be either a limit on the maximum number of non-improving iterations, or a time limit T.

Basic VLS operates by alternating between two distinct phases: one focused on shaking within the data dimension $\mathcal{N}^1$ $(i = 1)$, and the other on shaking within the dimension of problem formulation $\mathcal{N}^2$ $(i = 2)$. Each phase $i \in \{1, 2\}$ sets its own limit on the maximum number of iterations within the phase: $S_1 \in \mathbb{N}$ and $S_2 \in \mathbb{N}$, respectively. For convenience, these two integer numbers are combined into a single vector $S = (S_1, S_2)$.

Also, the minimum and maximum values of the shake parameter k are unique for each phase. The values stored in the matrix $K \in \mathbb{R}^{2\times 2}$ are used to define the admissible ranges of shaking power k in the data and formulation phases, respectively.

Before local search can be applied in the perturbed landscape L', the incumbent solution x should be translated onto the feasible solution space of L' using operator $T_{L,L'}$. The set of operators $\{T_{L,L'}\}_{L,L'\in\mathbb{L}}$ governing these transitions should be defined beforehand.

The pseudocode for Basic Variable Landscape Search (BVLS) is given in Algorithm 32. For initialization, the following starting values can be used:

- $L \leftarrow$ Select a current landscape from $\mathbb{L}$ (optionally at random);
- $x \leftarrow$ Select an initial current feasible solution from the feasible region S of landscape L (optionally at random).

It is worth noting that the basic VLS framework (Algorithm 32) may well use an alternative strategy to switch between phases: thoroughly exploiting one phase before moving to another. This can be achieved by changing the condition of the inner loop at line 7 of Algorithm 32 to checking if the maximum number of non-improving iterations has been exceeded within the current phase instead of the total number of iterations. However, the current version of VLS presented in Algorithm 32 aims to strike a balance between phase

Algorithm 32: Basic Variable Landscape Search (BVLS)

```
1  Function VLS(x, L, 𝓛, K, S, T):
2      t ← 0;
3      i ← 0;
4      while t < T do
5          k ← K^i_min;
6          s_i ← 0;
7          while s_i < S_i do
8              L' ← Shake_landscape(L, 𝓛, k, i);
9              x ← T_{L,L'}(x);
10             x' ← Local_search(x, L');
11             Neighborhood_change_sequential(x, x', L, L', k, K^i_min, K^i_max);
12             s_i ← s_i + 1;
13             t ← t + 1;
14         end
15         i ← (i + 1) mod 2;
16     end
17 return x
```

switching and exploitation by choosing the appropriate bounds S_i on the iteration counter for each phase i. This approach can be viewed as a kind of natural shaking arising from phase changing.

Another way to speed up the metaheuristic, increase its effectiveness and achieve a natural balance in exploiting different phases can be to distribute VLS jobs between multiple parallel workers in some proportion. Then, either a competitive, cooperative, or hybrid scheme can be used for communication between the parallel workers [MM23]. Determining the effectiveness and efficiency of these kinds of parallel settings constitutes a promising future research direction.

Incorporating a mechanism to adaptively change the objective landscape, VLS can be classified as a metaheuristic method, given that it embodies all the core elements of metaheuristic optimization strategies. VLS harmonizes local search (intensive search around the current area in the solution space) and global search (extensive search across the entire solution spaces of various landscapes), in order to escape from local optima and reach the global

optimum.

Within the domain of optimization, the Variable Formulation Search (VFS) method shares several characteristics with the proposed Variable Landscape Search (VLS) approach. However, it is crucial to underscore that VFS is in fact a special case nested within the more encompassing and versatile VLS method proposed in this article.

VFS accomplishes variability in the solution space by facilitating a broadened exploration of potential solutions across diverse problem formulations. This method focuses on exploring different problem representations, objective functions, and constraint sets to create a dynamic landscape of diverse solutions.

Nonetheless, the VLS framework takes this concept a step further. In addition to variable problem formulations, the VLS approach incorporates the modification of the task's input data (datasets) themselves. This extra dimension of flexibility allows for dynamic alteration of the landscape's search space, thereby introducing a higher degree of variability and resilience into the solution search process.

Consequently, it is accurate to posit that the VLS approach is a generalization of the VFS methodology. VLS encapsulates the benefits of variable problem formulation inherent in VFS and extends them to include variations in the input data. The resultant enhanced navigability through a larger, more diversified solution landscape heightens the potential for identifying superior solutions and delivering robust optimization results. Hence, the proposed VLS methodology offers a substantial leap forward in the science of optimization, exhibiting higher adaptability and versatility compared to its predecessors.

The Variable Landscape Search (VLS) metaheuristic stands out among other optimization techniques due to its unique approach of directly and adaptively manipulating the search space itself during the search process. This is a distinctive aspect, as most traditional metaheuristics operate within a fixed search space and focus on manipulating the search process, such as modifying the trajectory, introducing randomness, or adaptively tuning parameters.

Here are some aspects that make VLS unique:

1. *Dynamic / Variable Solution Space:* In most metaheuristics, the search space is predefined and remains static throughout the search process. In contrast, VLS changes the search space iteratively, which introduces a higher degree of dynamism and adaptability;

2. *Balance between Exploration and Exploitation:* The VLS algorithm's systematic manipulation of the objective function landscape helps to strike a better balance between exploration (searching new areas of the solution space) and exploitation (refining current promising solutions). This is achieved by altering the search space itself, which could help avoid entrapment in local minima and promote exploration of previously inaccessible regions of the solution space;

3. *Robustness:* The VLS approach can enhance the robustness of the search process, as the changing search space can provide different perspectives and opportunities to escape from sub-optimal regions. This could potentially lead to more consistent and reliable optimization results;

4. *Adaptability:* The VLS is not tied to a specific problem domain. The concept of varying the objective landscape can be applied to a broad range of optimization problems, making VLS a flexible and adaptable metaheuristic.

5. *Complexity Management:* By actively managing the search space, VLS might provide a novel approach to deal with high-dimensional and complex optimization problems where traditional methods struggle;

6. *Opportunity for Hybridization:* VLS provides an opportunity for hybridization with other metaheuristics, potentially leading to more powerful and efficient algorithms.

In summary, the VLS approach provides a fresh perspective on optimization, offering potential benefits in terms of robustness, adaptability, and ability to manage complex prob-

lem domains. This novel approach to directly and adaptively manipulate the solution space could open new avenues in the design and application of metaheuristics.

8.8 Examples

8.8.1 Big-means Algorithm

The Big-means algorithm (Chapter 3) uses the Minimum Sum-of-Squares Clustering (MSSC) formulation (1.1). To avoid ambiguities in notation, in this chapter we use letter p to denote the number of clusters, X to denote a sample, and $\mathbb{X}$ to denote the whole given input dataset.

In Big-means, we restrict ourselves only to the MSSC formulation (1.1). Then, the space $\mathbb{L}$ of all objective landscapes takes the following form:

$$\mathbb{L} = \{ \mathscr{L}(X, \text{MSSC}) \mid X \subseteq \mathbb{X} \}$$

The main idea of the Big-means algorithm and its various improved versions is to follow the stochastic approach, according to which the true $\mathbb{L}$ can be reasonably replaced by its stochastic version:

$$\widetilde{\mathbb{L}} = \{ \mathscr{L}(X^s, \text{MSSC}) \mid X^s \sim \mathcal{U}(\mathbb{X}, l),\ s = 0, \ldots, |\mathbb{X}| \}$$

where $\mathcal{U}(\mathbb{X}, s)$ denotes the distribution of s-sized samples drawn uniformly at random from $\mathbb{X}$. Thus, the search space of each landscape $L = \mathscr{L}(X^s, \text{MSSC}) \in \mathbb{L}$ consists of all locally optimal feasible solutions C for the following MSSC optimization problem defined on sample X^s:

$$\min_{C} \quad f\,(C, X^s) = \sum_{x \in X^s} \min_{j=1,\ldots,p} \|x - c_j\|^2 \tag{8.5}$$

The Big-means paradigm [Mus+23] further restricts the space $\mathbb{L}$ for an appropriate fixed range of the sample size s. The hope is that the resulting objective landscapes would have search spaces reasonably approximating the search space of the original landscape $L^* =$

$\mathscr{L}(\mathbb{X}, \text{MSSC})$:

$$\widetilde{\mathbb{L}}^{[s_{\min}, s_{\max}]} = \{\mathscr{L}(X^s, \text{MSSC}) \mid X^s \sim \mathcal{U}(\mathbb{X}, s),\ s = s_{\min}, \ldots, s_{\max}\}$$

Specifically, the assumption is that by properly and swiftly combining promising local optima of objective landscapes in $\widetilde{\mathbb{L}}^{[s_{\min}, s_{\max}]}$, one can obtain a close approximation to the global optima of the original objective landscape L^*.

Then, the following neighborhood structure can defined on the corresponding data space:

$$\mathcal{N}_k^1\left(X^s\right) = \left\{X^{s'} \sim \mathcal{U}(\mathbb{X}, s') \mid s_{\min} \leq s' \leq s_{\max},\ |s - s'| \leq k\right\},\ k = K^1_{\min}, \ldots, K^1_{\max} \quad (8.6)$$

The used absolute value distance metric in (8.6) clearly satisfies the required monotonicity property (8.3), producing a sequence of embedded neighborhoods due to the property that uniform samples of larger sizes subsume uniform samples of smaller sizes.

Moreover, the original Big-means algorithm [Mus+23] posits that it might be enough to assume $s_{\min} = s_{\max} = s$ for some appropriate fixed choice of the sample size $s = 0, \ldots, m$ with $s \ll m$:

$$\widetilde{\mathbb{L}}^s = \{\mathscr{L}(X^s, \text{MSSC}) \mid X^s \sim \mathcal{U}(\mathbb{X}, s)\}$$

Since algorithms based on the Big-means paradigm consider only one problem formulation (MSSC), they do not employ any neighborhood structure on the formulation space.

Various Big-means-based algorithms fall under the following cases of the general VLS framework (Algorithm 32):

1. *Big-means* [Mus+23] uses the K-means local search procedure and the sequential neighborhood change (Algorithm 31) with the following choice of parameters:

$$s_{\min} = s_{\max} = s \ll m$$

$$K^1_{\min} = K^1_{\max} = 0$$

$$S_1 = 1$$

$$S_2 = 0$$

2. *BigOptimaS3* maintains several parallel workers, each of which follows the basic VLS scheme, using the K-means local search procedure and the sequential neighborhood change (Algorithm 31) with the following choice of parameters:

$$1 \leq s_{\min} \leq s_{\max} \leq m$$

$$K^1_{\min} = K^1_{\max} = \infty \text{ (for the first iteration in a phase)}$$
$$K^1_{\min} = K^1_{\max} = 0 \text{ (for the remaining iterations in the phase)}$$

$$S_1 \in \mathbb{N}$$

$$S_2 = 0$$

Also, BigOptimaS3 adds a special procedure to end the search process. It picks the landscape sample size s_{opt} that is most likely to yield improvement in the objective function value based on the collected history of improving samples sizes across iterations. Then, BigOptimaS3 realizes the optimal landscape using s_{opt} and selects the centroids of the parallel worker giving the best result in the optimal landscape;

3. *BigVNSClust* uses the K-means local search procedure and the sequential neighborhood change (Algorithm 31) with the same choice of VLS parameters as Big-means. Additionally, BigVNSClust adds an extra shaking dimension that acts around the incumbent solution within the feasible solution space of every shaken landscape. This kind of shaking happens simultaneously with the input data shaking. In each iteration, BigVNSClust uses the cyclic neighborhood change scheme to transition from one neighborhood structure to another within the current feasible region, which cyclically increments the shaking power parameter in every iteration irrespective of whether the improvement has taken place or not.

In all these algorithms, the local search within the initial feasible solution space S in Algorithm 32 is performed by starting from the distribution of points defined by K-means++ on S, while the incumbent solution x (C in the context of MSSC) is used to initialize the local search procedure in all the subsequent iterations.

The conceptualization and execution of the Big-means algorithm are steeped in fundamental optimization principles. The approach encapsulates and harnesses the power of iterative exploration and manipulation of the search space, overcoming the traditional challenges associated with local minima, and relentlessly pursuing the global optimum.

The perturbation introduced into the clustering results through the "shaking" procedure is a pivotal aspect of the Big-means algorithm. Each iteration yields a new sample, creating variability and diversity in the centroid configurations. Each sample serves as a sparse approximation of the full data cloud in the n-dimensional feature space. This stochastic sampling offers a pragmatic balance between exhaustive search (which is often computationally infeasible) and deterministic search (which risks entrapment in local minima). By including a diverse set of sparse approximations, the solution space exploration becomes more robust, adaptive, and capable of reaching the global optimum.

The Big-means algorithm is inherently an adaptive search strategy. Instead of maintaining a fixed search space, it allows the search space to evolve dynamically by sampling different parts of the data cloud in each iteration. This leads to an "adaptive landscape", a powerful metaphor where the distribution of locally optimal solutions (the search space) can evolve over the course of optimization, much like species evolving in a changing environment.

This dynamism is beneficial on two fronts: it assists in avoiding the entrapment in local optima, and it promotes a robust exploration of the solution space. If the algorithm is stuck in a local optimum with a given sample, a new random sample might change the landscape such that the local optimum becomes a hill, and better solutions appear as valleys.

The visual representation in Figure 8.1 can be easily interpreted as Big-means iterations, where x is the incumbent set of centroids C, and each new landscape $f_i(C)$ is the MSSC objective function restricted to a new sample $f(C, X_i)$. This figure is a testament to the Big-

means algorithm's underlying principle of systematically exploring and manipulating the search space. The interplay of randomness (through sampling) and determinism (through local optimization) in the algorithm provides a potent strategy to tackle the notorious problem of local minima in clustering algorithms.

8.8.2 Variable Formulation Search (VFS)

The idea of VFS proposed in [Par+13] retains the same steps as the basic VNS metaheuristic [Bri+23] except for using the special `Accept(x,x',r)` procedure in all of them, where x is an incumbent solution, x' is a candidate solution, and r defines a range of considered formulations $\mathbb{F} = \{F_1, \ldots, F_r\}$ and their corresponding objective functions $\{f_1, \ldots, f_r\}$. The `Accept` procedure is listed in Algorithm 33. This procedure checks if the candidate solution x' leads to improvements in the objective functions by iteratively proceeding to the next formulation in case of a tie in the current one and rejecting x' as long as it causes a decrease in any of them. This approach is effective in tackling the issue of a flat landscape. This implies that, when a single formulation of an optimization problem is employed, a large number of solutions that are close to a given solution tend to have identical values in terms of their objective function. This scenario makes it challenging to identify which nearby solution is the better option for further exploration in the search process. However, our VLS metaheuristic subsumes this VFS idea by simply including the `Accept` procedure into the local search and neighborhood change steps.

The concept of Formulation Space Search (FSS), as discussed in Mladenovic's study [MPU07], can be considered a specific instance within the broader framework of the Variable Landscape Search (VLS) metaheuristic. This interpretation becomes apparent when you observe that FSS represents what happens when VLS is confined to the single "formulation" phase. Essentially, FSS is a special case of VLS where the process is limited to just the formulation aspect.

The idea of Variable Search Space (VSS) proposed in [HPZ08] also falls under the VLS metaheuristic. One can check that by restricting to the "formulation" phase, modifying the

Algorithm 33: Accept procedure

```
1  Procedure Accept(x, x′, r):
2      for i = 0 to r do
3          condition1 ← f_i(x′) < f_i(x);
4          condition2 ← f_i(x′) > f_i(x);
5          if condition1 then
6              return True;
7          else if condition2 then
8              return False;
9          else
10             continue; // with the next alternative formulation
11         end
12     end
13     return False
```

neighborhood change step to the cyclic one, and defining the neighborhood structure on the formulation space as

$$\mathcal{N}_k^2 = \{F_k\}$$

where $K^2_{\min} = 1$ and $K^2_{\max} = r$ are the shake parameters, and $\mathbb{F} = \{F_1, \ldots, F_r\}$ is a finite set of r available formulations for the underlying optimization problem.

8.9 Conclusion and future research

This chapter puts forth the novel Variable Landscape Search (VLS) metaheuristic approach that actively manipulates the search space itself during optimization. Unlike traditional methods that use a fixed, static objective function landscape and its search space, VLS dynamically adjusts them over iterations. This flexibility stems from incorporating changes in the problem formulation as well as modifications of the input data. By expanding the scope of exploration, VLS facilitates the discovery of superior solutions. The algorithm balances focused local search with extensive global search across evolving formulations and input datasets. This helps overcome poor local optima that restrict traditional techniques. The conceptualization of the Big-means clustering algorithm, as well as its various more

advanced versions and VFS, under the VLS lens offers insights into how shifting objective landscapes assists in bypassing local minima. The unique strategy and broad applicability of VLS across problem domains position it as a versatile addition to the metaheuristic toolkit for global optimization.

An intriguing avenue for future research lies in the enhancement of the VLS metaheuristic by incorporating a shaking mechanism for the incumbent solution within its respective landscape. Such an addition, conceptualized as a distinct phase in the VLS process, aims to further refine and diversify the search strategy. This proposed enhancement has the potential to significantly increase the robustness and efficacy of the VLS metaheuristic in exploring complex search spaces.

Chapter 9

CONCLUSION

In this dissertation, we conducted an extensive exploration of the application of K-means clustering to big data, examining the problem from multiple critical facets: scalability, decomposition, accuracy, computational efficiency, simplicity, parallelizability, adaptive parameter selection, and the incorporation of advanced metaheuristic techniques. This comprehensive approach has successfully addressed all the research questions posited at the outset of this study.

- RQ 1: What are the limitations of traditional and alternative clustering algorithms in analyzing big datasets? How can the traditional K-means algorithm be optimized in general and specifically for big data?

  The exploration of these limitations is detailed in Chapters 3 and 5. It was found that both traditional and alternative clustering algorithms exhibit suboptimal performance in comparison to the Big-means algorithm. These chapters provide a comprehensive discussion on the reasons behind this difference in performance, delving into the specific inadequacies and constraints of the traditional and alternative methods when applied to large-scale data analysis. A detailed study of various K-means optimization strategies has also been conducted;

- RQ 2: Is there a clustering method that concurrently optimizes key criteria such as accuracy, speed, and simplicity (as per LIMA dominance [Bri+23]) when applied to big data? If not, can a novel approach be developed and empirically validated to maximize these criteria in big data clustering?

These queries are addressed in Chapters 3 and 5. The Big-means method emerges as a superior technique, demonstrating its capability to concurrently optimize all three aforementioned criteria. This conclusion is substantiated by extensive experimental evaluations employing real-world datasets, which underscore its efficacy and applicability in practical big data scenarios;

- RQ 3: Does the widely accepted belief that utilizing every data point in a dataset is essential for attaining a satisfactory clustering solution hold true?

  The findings presented in Chapters 3, 4, 6, and 7 challenge this prevailing belief by introducing innovative state-of-the-art methodologies. These methodologies leverage the decomposition principle, demonstrating that it is feasible to achieve high levels of accuracy and efficiency without the necessity of incorporating every data point;

- RQ 4: Does the prevailing trend of relying on more complex hybrid approaches hold true in the quest for improved clustering solutions?

  Insights into this question are presented in Chapter 5. The findings, as detailed in Tables 5.2 and 5.2, indicate that the prevailing trend towards complexity may not be necessary. Notably, algorithms that achieve the optimal balance between accuracy and speed tend to have lower LIMA numbers, reflecting a greater degree of simplicity. This challenges the assumption that complexity is synonymous with effectiveness in clustering algorithms;

- RQ 5: Based on the above findings, can we establish the precise definition of a "true big data" clustering algorithm?

  Reflecting on the analysis in Chapter 3, a "true big data" clustering algorithm can be defined as one that exemplifies an optimal balance among three critical dimensions: simplicity in its operational framework, high quality of results, and rapid convergence speed. Additionally, it distinguishes itself by conducting a global search for solutions

specifically tailored to big data contexts, without the dependency on complex global optimization metaheuristics. Key attributes of such an algorithm include its scalability, manifested through features like configurable quality-speed trade-offs, consistent effectiveness regardless of data size variations, enhanced result quality proportional to data volume, proficient processing of partial datasets, and advanced capabilities for parallel, distributed, and stream processing;

- RQ 6: How can a "true big data" clustering algorithm be optimized to make the most effective use of the modern high-performance computing (HPC) technologies?

  An in-depth exploration of this topic is presented in Chapter 4. This chapter delves into the various parallelization schemes applicable to decomposition-based big data clustering algorithms, encompassing inner, competitive, cooperative, and hybrid parallelization strategies. Each scheme is scrutinized for its potential to enhance the efficiency and effectiveness of "true big data" clustering algorithms when integrated with advanced HPC technologies, thereby maximizing computational performance and algorithmic scalability;

- RQ 7: How can a "true big data" clustering algorithm be further enhanced by incorporating modern metaheuristic optimization techniques?

  Chapter 6 addresses this question. It illustrates how the Big-means algorithm can be significantly enhanced through the incorporation of the Variable Neighborhood Search (VNS) metaheuristic. This chapter elaborates on the methodology behind this integration and evaluates the impact of VNS on the performance, efficiency, and accuracy of Big-means in handling large datasets;

- RQ 8: Could the accuracy and efficiency of Big-means be further boosted by applying stochastic sample size optimization within a competitive parallel framework?

  Chapter 7 presents a detailed investigation into this possibility. The findings reveal

that the integration of stochastic sample size optimization within a competitive parallel framework contributes significantly to the enhancement of the Big-means algorithm's performance, eliminating the need for manual sample size selection;

- RQ 9: Can a solid theoretical foundation be established for the proposed algorithms in the form of a metaheuristic that is comprehensive, practically applicable, and encompasses other existing metaheuristics in the literature?

  Chapter 8 introduces a novel metaheuristic framework that mathematically extends and generalizes the concept of utilizing the input data decomposition principle in the context of big data clustering challenges. This innovative approach not only integrates the algorithms proposed in earlier chapters but also establishes a unifying mathematical foundation that aligns with several other existing metaheuristics in the current literature. This comprehensive formulation provides a broader perspective and a more cohesive understanding of how the data decomposition principle can be effectively applied across various big data clustering scenarios, bridging gaps and offering new insights into algorithmic development and optimization;

Significantly, our research has led to the development of four innovative algorithms for big data clustering. Each of these algorithms, in their turn, advanced the field, establishing themselves as the state of the art. Additionally, our work includes an exhaustive review of existing big data clustering methodologies, uniquely assessed through the "less is more" lens. The dual approach — both empirical, with extensive experiments on diverse real-world datasets, and theoretical, through the development of a novel general optimization metaheuristic — provides a robust foundation for our findings.

Looking forward, we are enthusiastic about pursuing the myriad of promising research avenues suggested in the conclusion sections of each chapter. This work has not only contributed to the field of data clustering but has also opened new pathways for future exploration and innovation.

BIBLIOGRAPHY

[Hol14] A. Holder, ed. *Mathematical Programming Glossary*. Originally authored by Harvey J. Greenberg, 1999-2006. http://glossary.computing.society.informs.org: INFORMS Computing Society, 2006–14.

[Bri+23] Jack Brimberg, Said Salhi, Raca Todosijević, and Dragan Urošević. "Variable Neighborhood Search: The power of change and simplicity". In: *Computers & Operations Research* 155 (2023), p. 106221. ISSN: 0305-0548. DOI: 10.1016/j.cor.2023.106221.

[GJN23] Giorgio Giorgi, Bienvenido Jiménez, and Vicente Novo. "Sensitivity Analysis". In: *International Series in Operations Research and Management Science* 344 (2023), pp. 225–242. DOI: 10.1007/978-3-031-30324-1_7.

[HLD23] Jiawei Huang, Wenjie Liu, and Hu Ding. *Is Simple Uniform Sampling Effective for Center-Based Clustering with Outliers: When and Why?* 2023. arXiv: 2103.00558 [cs.LG].

[MM23] Ravil Mussabayev and Rustam Mussabayev. *Strategies for Parallelizing the Big-Means Algorithm: A Comprehensive Tutorial for Effective Big Data Clustering*. 2023. arXiv: 2311.04517 [cs.DC].

[Mus+23] Rustam Mussabayev, Nenad Mladenovic, Bassem Jarboui, and Ravil Mussabayev. "How to Use K-means for Big Data Clustering?" In: *Pattern Recognition* 137 (2023), p. 109269. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2022.109269. URL: https://www.sciencedirect.com/science/article/pii/S0031320322007488.

[VS23] Leonardo Vanneschi and Sara Silva. “Optimization Problems and Local Search”. In: *Natural Computing Series* (2023), pp. 13–44. DOI: 10.1007/978-3-031-17922-8_2.

[APK22] Anis Alazzawe, Amitangshu Pal, and Krishna Kant. “Efficient Big-Data Access: Taxonomy and a Comprehensive Survey”. In: *IEEE Transactions on Big Data* 8.2 (2022). Cited by: 2; All Open Access, Bronze Open Access, pp. 356–376. DOI: 10.1109/TBDATA.2020.3036813.

[GRV22] A. J. Gallego, J. R. Rico-Juan, and J. J. Valero-Mas. “Efficient k-nearest neighbor search based on clustering and adaptive k values”. In: *Pattern Recognition* (2022). DOI: 10.1016/j.patcog.2021.108356.

[KBD22] P. Kalczynski, J. Brimberg, and Z. Drezner. “Less is more: simple algorithms for the minimum sum of squares clustering problem”. In: *IMA Journal of Management Mathematics* (2022). DOI: 10.1093/imaman/dpab031.

[MTL+22] W. Ma, X. Tu, B. Luo, et al. “Semantic clustering based deduction learning for image recognition and classification”. In: *Pattern Recognition* (2022). DOI: 10.1016/j.patcog.2021.108440.

[MPP22] N. Mladenovic, J. Pei, and P. Pardalos. “Less is more approach in optimization: a road to artificial intelligence”. In: *Optimization Letters* (2022). DOI: 10.1007/s11590-021-01818-w.

[MBU22] Nenad Mladenović, Jack Brimberg, and Dragan Urošević. *Formulation space search metaheuristic*. 2022, pp. 405–445. DOI: 10.1007/978-3-030-96935-6_12.

[MS22] Fatemeh Moodi and Hamid Saadatfar. “An improved K-means algorithm for big data”. In: *IET Software* 16.1 (2022), pp. 48–59. DOI: 10.1049/sfw2.12032.

[PK22] Dhidhi Pambudi and Masaki Kawamura. “Constructing the Neighborhood Structure of VNS Based on Binomial Distribution for Solving QUBO Problems”. In: *Algorithms* 15.6 (2022). ISSN: 1999-4893. DOI: 10.3390/a15060192. URL: https://www.mdpi.com/1999-4893/15/6/192.

[PSW22] V. Piccialli, A. M. Sudoso, and A. Wiegele. “SOS-SDP: An Exact Solver for Minimum Sum-of-Squares Clustering”. In: *INFORMS Journal on Computing* (2022). DOI: 10.1287/ijoc.2022.1166.

[PBB22] Christos Psarras, Henrik Barthels, and Paolo Bientinesi. “The Linear Algebra Mapping Problem. Current State of Linear Algebra Languages and Libraries”. In: *ACM Trans. Math. Softw.* 48.3 (Sept. 2022). ISSN: 0098-3500. DOI: 10.1145/3549935.

[AMM21] I. Akhmetov, R. Mussabayev, and N. Mladenovic. “Using K-Means and Variable Neighborhood Search for Automatic Summarization of Scientific Articles”. In: *International Conference on Variable Neighborhood Search* (2021). DOI: 10.1007/978-3-030-69625-2_13.

[AAS21] Rasim M. Alguliyev, Ramiz M. Aliguliyev, and Lyudmila V. Sukhostat. “Parallel batch k-means for Big data clustering”. In: *Computers & Industrial Engineering* (2021). DOI: 10.1016/j.cie.2020.107023.

[CGa21] Vincent Cohen-Addad, Benjamin Guedj, and et al. “Online k-means clustering”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 2536–2544.

[CDC21] Erik Cuevas, Primitivo Diaz, and Octavio Camarena. “Experimental analysis between exploration and exploitation”. In: *Intelligent Systems Reference Library* 195 (2021), pp. 249–269. DOI: 10.1007/978-3-030-58100-8_10.

[DLS21] Zineb Dafir, Yasmine Lamari, and Said Chah Slaoui. "A survey on parallel clustering algorithms for Big Data". In: *Artificial Intelligence Review* 54.4 (Apr. 2021), pp. 2411–2443. ISSN: 1573-7462. DOI: 10.1007/s10462-020-09918-2.

[Dje+21] Youcef Djenouri, Asma Belhadi, Djamel Djenouri, and Jerry Chun-Wei Lin. "Cluster-based information retrieval using pattern mining". In: *APPLIED INTELLIGENCE* 51.4 (Apr. 2021), pp. 1888–1903. ISSN: 0924-669X. DOI: 10.1007/s10489-020-01922-x.

[KMM21] Olzhas Kozbagarov, Rustam Mussabayev, and Nenad Mladenovic. "A New Sentence-Based Interpretative Topic Modeling and Automatic Topic Labeling". In: *Symmetry* (2021). DOI: 10.3390/sym13050837.

[LG21] Humberto Elias Garcia Lopes and Marlusa de Sevilha Gosling. "Cluster Analysis in Practice: Dealing with Outliers in Managerial Research; [Análise de Clusters na Prática: Lidando com Outliers na Pesquisa Gerencial]". In: *Revista de Administracao Contemporanea* 25.1 Special Issue (2021). All Open Access, Gold Open Access, Green Open Access. DOI: 10.1590/1982-7849rac2021200081.

[MHE21a] Mahmoud A. Mahdi, Khalid M. Hosny, and Ibrahim Elhenawy. "Scalable Clustering Algorithms for Big Data: A Review". In: *IEEE Access* (2021). DOI: 10.1109/ACCESS.2021.3084057.

[MHE21b] Mahmoud A. Mahdi, Khalid M. Hosny, and Ibrahim Elhenawy. "Scalable Clustering Algorithms for Big Data: A Review". In: *IEEE Access* 9 (2021). All Open Access, Gold Open Access, pp. 80015–80027. DOI: 10.1109/ACCESS.2021.3084057.

[MS21] P. Mansueto and F. Schoen. "Memetic differential evolution methods for clustering problems". In: *Pattern Recognition* (2021). DOI: 10.1016/j.patcog.2021.107849.

[Mit+21] Himanshu Mittal, Avinash Chandra Pandey, Raju Pal, and Ashish Tripathi. "A new clustering method for the diagnosis of CoVID19 using medical images". In: *APPLIED INTELLIGENCE* 51.5, SI (May 2021), pp. 2988–3011. ISSN: 0924-669X. DOI: 10.1007/s10489-020-02122-3.

[Muk+21] Ravil I. Mukhamediev, Adilkhan Symagulov, Yan Kuchin, Kirill Yakunin, and Marina Yelis. "From Classical Machine Learning to Deep Neural Networks: A Simplified Scientometric Review". In: *Applied Sciences* (2021). DOI: 10.3390/app11125541.

[SM21] J. Saha and J. Mukherjee. "CNAK: Cluster number assisted K-means". In: *Pattern Recognition* (2021). DOI: 10.1016/j.patcog.2020.107625.

[CPL20] Marco Capo, Aritz Perez, and Jose A. Lozano. "An efficient K-means clustering algorithm for tall data". In: *Data Mining and Knowledge Discovery* (2020). DOI: 10.1007/s10618-020-00678-9.

[CYY20] T. H. Cuong, J. Yao, and N. D. Yen. "Qualitative properties of the minimum sum-of-squares clustering problem". In: *A Journal of Mathematical Programming and Operations Research* (2020). DOI: 10.1080/02331934.2020.1778685.

[DLS20] Zineb Dafir, Yasmine Lamari, and Said Chah Slaoui. "A survey on parallel clustering algorithms for Big Data". In: *PeerJ Computer Science* (2020). DOI: 10.1007/s10462-020-09918-2.

[Dra+20] John H. Drake, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. "Recent advances in selection hyper-heuristics". In: *European Journal of Operational Research* 285.2 (2020). All Open Access, Green Open Access, Hybrid Gold Open Access, pp. 405–428. DOI: 10.1016/j.ejor.2019.07.073.

[Hua+20] Dong Huang, Chang-Dong Wang, Jian-Sheng Wu, Jian-Huang Lai, and Chee-Keong Kwoh. "Ultra-Scalable Spectral Clustering and Ensemble Clustering".

English. In: *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING* 32.6 (June 2020), pp. 1212–1226. ISSN: 1041-4347. DOI: 10.1109/TKDE.2019.2903410.

[KMM20] Alexander Krassovitskiy, Nenad Mladenovic, and Rustam Mussabayev. "Decomposition/Aggregation K-means for Big Data". In: *International Conference on Mathematical Optimization Theory and Operations Research* (2020). DOI: 10.1007/978-3-030-58657-7_32.

[MMV20] Alguliyev R. M., Aliguliyev R. M., and Sukhostat L. V. "Efficient algorithm for big data clustering on single machine". In: *CAAI Transactions on Intelligence Technology* (2020). DOI: 10.1049/trit.2019.0048.

[MRS20] Konstantin Makarychev, Aravind Reddy, and Liren Shan. "Improved Guarantees for K-means++ and K-means++ Parallel". In: *Advances in Neural Information Processing Systems* (2020).

[SAA20] Mozamel M. Saeed, Zaher Al Aghbari, and Mohammed Alsharidah. "Big data clustering techniques based on Spark: a literature review". In: *PeerJ Computer Science* (2020). DOI: 10.7717/peerj-cs.321.

[SJB20] M. Selosse, J. Jacques, and C. Biernacki. "Textual data summarization using the Self-Organized Co-Clustering model". In: *Pattern Recognition* (2020). DOI: 10.1016/j.patcog.2020.107315.

[Tu+20] Bing Tu, Xianchang Yang, Nanying Li, Chengle Zhou, and Danbing He. "Hyperspectral anomaly detection via density peak clustering". In: *PATTERN RECOGNITION LETTERS* 129 (Jan. 2020), pp. 144–149. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2019.11.022.

[WWG20] Sen Wu, Yu-zhi Wang, and Xiao-nan Gao. "Clustering algorithm for imbalanced data based on nearest neighbor". In: *Gongcheng Kexue Xuebao/Chinese*

Journal of Engineering 42.9 (2020). Cited by: 4, pp. 1209–1219. DOI: 10.13374/j.issn2095-9389.2019.10.09.003.

[XNH20] Dong Xia, Fan Ning, and Wei He. "Research on Parallel Adaptive Canopy-K-Means Clustering Algorithm for Big Data Mining Based on Cloud Platform". In: *Journal of Grid Computing* 18.2 (2020), pp. 263–273. DOI: 10.1007/s10723-019-09504-z.

[AAB19] A. Adolfsson, M. Ackerman, and N. C. Brownstein. "To cluster, or not to cluster: An analysis of clusterability methods". In: *Pattern Recognition* (2019). DOI: 10.1016/j.patcog.2018.10.026.

[Alg+19] Rasim M. Alguliyev, Ramiz M. Aliguliyev, Nijat R. Isazade, Asad Abdi, and Norisma Idris. "COSUM: Text summarization based on clustering and optimization". In: *EXPERT SYSTEMS* 36.1 (Feb. 2019). ISSN: 0266-4720. DOI: 10.1111/exsy.12340.

[BBE19] Mohamed Aymen Ben HajKacem, Chiheb-Eddine Ben N'Cir, and Nadia Essoussi. "STiMR k-Means: An Efficient Clustering Method for Big Data". In: *International Journal of Pattern Recognition and Artificial Intelligence* 33.08 (2019), p. 1950013. DOI: 10.1142/S0218001419500137.

[DS19] Marco Dorigo and Thomas Stützle. "Ant colony optimization: Overview and recent advances". In: *International Series in Operations Research and Management Science* 272 (2019), pp. 311–351. DOI: 10.1007/978-3-319-91086-4_10.

[DAM19] D. Dzamic, D. Aloise, and N. Mladenovic. "Ascent–descent variable neighborhood decomposition search for community detection by modularity maximization". In: *Annals of Operations Research volume* (2019). DOI: 10.1007/s10479-017-2553-9.

[FS19] Pasi Franti and Sami Sieranoja. "How much can k-means be improved by using better initialization and repeats?" In: *Pattern Recognition* (2019). DOI: 10.1016/j.patcog.2019.04.014.

[GP19] Michel Gendreau and Jean-Yves Potvin. "Tabu Search". In: *International Series in Operations Research and Management Science* 272 (2019), pp. 37–55. DOI: 10.1007/978-3-319-91086-4_2.

[GMS19] Manfred Gilli, Dietmar Maringer, and Enrico Schumann. *Numerical Methods and Optimization in Finance*. 2nd ed. Academic Press, 2019. DOI: 10.1016/C2017-0-01621-X.

[GV19] Daniel Gribel and Thibaut Vidal. "HG-means: A scalable hybrid genetic algorithm for minimum sum-of-squares clustering". In: *Pattern Recognition* (2019). DOI: 10.1016/j.patcog.2018.12.022.

[LMS19] Helena Ramalhinho Lourenço, Olivier C. Martin, and Thomas Stützle. "Iterated local search: Framework and applications". In: *International Series in Operations Research and Management Science* 272 (2019), pp. 129–168. DOI: 10.1007/978-3-319-91086-4_5.

[Ma+19] Xiaoliang Ma, Xiaodong Li, Qingfu Zhang, Ke Tang, Zhengping Liang, Weixin Xie, and Zexuan Zhu. "A Survey on Cooperative Co-Evolutionary Algorithms". In: *IEEE Transactions on Evolutionary Computation* 23.3 (2019), pp. 421–441. DOI: 10.1109/TEVC.2018.2868770.

[MM19] Gaurav Mishra and Sraban Kumar Mohanty. "A fast hybrid clustering technique based on local nearest neighbor using minimum spanning tree". In: *Expert Systems with Applications* 132 (2019), pp. 28–43. DOI: 10.1016/j.eswa.2019.04.048.

[Mla+19] Nenad Mladenovic, Abdulaziz Alkandari, Jun Pei, Raca Todosijevic, and Panos M. Pardalos. "Less is more approach: basic variable neighborhood search for the

obnoxious p-median problem". In: *International Transactions in Operational Research* (2019). DOI: 10.1111/itor.12646.

[NTM19] S. Ng, R. Tawiah, and G. J. McLachlan. "Unsupervised pattern recognition of mixed data structures with numerical and categorical features using a mixture regression modelling framework". In: *Pattern Recognition* (2019). DOI: 10.1016/j.patcog.2018.11.022.

[RR19] Mauricio G. C. Resende and Celso C. Ribeiro. "Greedy randomized adaptive search procedures: Advances and extensions". In: *International Series in Operations Research and Management Science* 272 (2019), pp. 169–220. DOI: 10.1007/978-3-319-91086-4_6.

[SBB+19] S. Seifollahi, A. Bagirov, E. Borzeshi, et al. "A simulated annealing-based maximum-margin clustering algorithm". In: *Computational Intelligence* (2019). DOI: 10.1111/coin.12187.

[ŚHT19] Marek Śmieja, Krzysztof Hajto, and Jacek Tabor. "Efficient mixture model for clustering of sparse high dimensional binary data". In: *Data Mining and Knowledge Discovery* 33.6 (2019). Cited by: 12; All Open Access, Green Open Access, Hybrid Gold Open Access, pp. 1583–1624. DOI: 10.1007/s10618-019-00635-1.

[YTZ+19] J. Yu, M. Tan, H. Zhang, et al. "Hierarchical Deep Click Feature Prediction for Fine-Grained Image Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019). DOI: 10.1109/TPAMI.2019.2932058.

[Aar+18] Catherine Aaron, Alejandro Cholaquidis, Ricardo Fraiman, and Badih Ghattas. "Multivariate and functional robust fusion methods for structured Big Data". In: *Journal of Multivariate Analysis* (2018). DOI: 10.1016/j.jmva.2018.06.012.

[BLK18] Olivier Bachem, Mario Lucic, and Andreas Krause. "Scalable k-Means Clustering via Lightweight Coresets". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018). DOI: 10.1145/3219819.3219973.

[Che+18] Xiaojun Chen, Yixiang Fang, Min Yang, Feiping Nie, Zhou Zhao, and Joshua Zhexue Huang. "PurTreeClust: A Clustering Algorithm for Customer Segmentation from Massive Customer Transaction Data". In: *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING* 30.3 (Mar. 2018), pp. 559–572. ISSN: 1041-4347. DOI: 10.1109/TKDE.2017.2763620.

[Dua+18] Abraham Duarte, Nenad Mladenović, Jesús Sánchez-Oro, and Raca Todosijević. *Variable neighborhood descent*. Vol. 1-2. 2018, pp. 341–367. DOI: 10.1007/978-3-319-07124-4_9.

[Ism18] Hassan Ismkhan. "I-k-means-+: An iterative clustering algorithm based on an enhanced version of the k-means". In: *Pattern Recognition* (2018). DOI: 10.1016/j.patcog.2018.02.015.

[KBT18] Napsu Karmitsa, Adil M. Bagirov, and Sona Taheri. "Clustering in large data sets with the limited memory bundle method". In: *Pattern Recognition* (2018). DOI: 10.1016/j.patcog.2018.05.028.

[KM18] Alexander Krassovitskiy and Rustam Mussabayev. "Energy-Based Centroid Identification and Cluster Propagation with Noise Detection". In: *Proceedings of the 10th Computational Collective Intelligence Conference* (2018). DOI: 10.1007/978-3-319-98443-8_48.

[LCR+18] Y. Lu, B. Cao, C. Rego, et al. "A Tabu search based clustering algorithm and its parallel implementation on Spark". In: *Applied Soft Computing* (2018). DOI: 10.1016/j.asoc.2017.11.038.

[Mar18] A. Marowka. "Python accelerators for high-performance computing". In: *The Journal of Supercomputing* 74.4 (2018), pp. 1449–1460.

[Moh+18] Amin Mohebi, Saeed Aghabozorgi, Teh Ying Wah, Tutut Herawan, and Ramin Yahyapour. "One-Shot Coresets: The Case of k-Clustering". In: *Artificial Intelligence and Statistics* (2018).

[SSG18] Kenneth Sörensen, Marc Sevaux, and Fred Glover. *A history of metaheuristics*. Vol. 2-2. 2018, pp. 791–808. DOI: 10.1007/978-3-319-07124-4_4.

[AB17] Suad A. Alasadi and Wesam S. Bhaya. "Review of data preprocessing techniques in data mining". In: *Journal of Engineering and Applied Sciences* 12.16 (2017), pp. 4102–4107. DOI: 10.3923/jeasci.2017.4102.4107.

[Alo+17] Daniel Aloise, Nielsen Castelo Damasceno, Nenad Mladenovic, and Daniel Nobre Pinheiro. "On Strategies to Fix Degenerate k-means Solutions". In: *Journal of Classification* (2017). DOI: 10.1007/s00357-017-9231-0.

[AH17] Charles Audet and Warren Hare. "Biobjective Optimization". In: *Springer Series in Operations Research and Financial Engineering* (2017), pp. 247–262. DOI: 10.1007/978-3-319-68913-5_14.

[BCL+17] L. Bai, X. Cheng, J. Liang, et al. "Fast density clustering strategies based on the k-means algorithm". In: *Pattern Recognition* (2017). DOI: 10.1016/j.patcog.2017.06.023.

[Bri+17] Jack Brimberg, Nenad Mladenovic, Raca Todosijevic, and Dragan Urosevic. "Less is more: solving the max-mean diversity problem with variable neighborhood search." In: *Information Sciences* (2017). DOI: 10.1016/j.ins.2016.12.021.

[DGK17] Akram Dehnokhalaji, Mojtaba Ghiyasi, and Pekka Korhonen. "Resource allocation based on cost efficiency". In: *Journal of the Operational Research Society* 68.10 (2017), pp. 1279–1289. DOI: 10.1057/s41274-016-0020-7.

[Die+17] Karl E. Dierckens, Adrian B. Harrison, Carson K. Leung, and Adrienne V. Pind. "A Data Science and Engineering Solution for Fast K-Means Clustering of Big Data". In: *2017 IEEE Trustcom/BigDataSE/ICESS*. 2017, pp. 925–932. DOI: 10.1109/Trustcom/BigDataSE/ICESS.2017.332.

[GRV17] A. J. Gallego, J. R. Rico-Juan, and J. J. Valero-Mas. "k-MS: a novel clustering algorithm based on morphological reconstruction". In: *Pattern Recognition* (2017). DOI: 10.1016/j.patcog.2016.12.027.

[Han+17] Pierre Hansen, Nenad Mladenović, Raca Todosijević, and Saïd Hanafi. "Variable neighborhood search: basics and variants". In: *EURO Journal on Computational Optimization* 5.3 (2017), pp. 423–454. ISSN: 2192-4406. DOI: 10.1007/s13675-016-0075-x.

[JMN17] Einollah Jafarnejad Ghomi, Amir Masoud Rahmani, and Nooruldeen Nasih Qader. "Load-balancing algorithms in cloud computing: A survey". In: *Journal of Network and Computer Applications* 88 (2017), pp. 50–71. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2017.04.007.

[KBT17] Napsu Karmitsaa, Adil M. Bagirov, and Sona Taheri. "New Diagonal Bundle Method for Clustering Problems in Large Data Sets". In: *Pattern Recognition* (2017). DOI: 10.1016/j.ejor.2017.06.010.

[Kra17] Oliver Kramer. *Genetic Algorithm Essentials*. 1st ed. Studies in Computational Intelligence. Springer Cham, Jan. 2017, pp. IX, 92. ISBN: 978-3-319-52155-8. DOI: 10.1007/978-3-319-52156-5.

[NMT17] A. Nikolaev, N. Mladenovic, and R. Todosijevic. "J-means and I-means for minimum sum-of-squares clustering on networks". In: *Optimization Letters* (2017). DOI: 10.1007/s11590-015-0974-4.

[PM17] Lalit M. Patnaik and Srinivas Mandavilli. *Adaptation in genetic algorithms*. 2017, pp. 45–64. DOI: 10.1201/9780203713402.

[Sax+17] Amit Saxena, Mukesh Prasad, Akshansh Gupta, Neha Bharill, Om Prakash Patel, Aruna Tiwari, Er Meng Joo, Ding Weiping, and Lin Chin-Teng. "A review of clustering techniques and developments". In: *Neurocomputing* (2017). DOI: 10.1016/j.neucom.2017.06.053.

[ZZZ17] Dan Zhang, Qi Zhu, and Daoqiang Zhang. "Multi-modal dimensionality reduction using effective distance". In: *Neurocomputing* 259 (2017). Cited by: 8, pp. 130–139. DOI: 10.1016/j.neucom.2016.07.075.

[BTU16] Adil M. Bagirov, Sona Taheri, and Julien Ugon. "Nonsmooth DC programming approach to the minimum sum-of-squares clustering problems". In: *Pattern Recognition* (2016). DOI: 10.1016/j.patcog.2015.11.011.

[HMT+16] P. Hansen, N. Mladenovic, R. Todosijevic, et al. "Variable neighborhood search: basics and variants". In: *EURO Journal on Computational Optimization* (2016). DOI: 10.1007/s13675-016-0075-x.

[HS16] Marwan Hassani and Thomas Seidl. "Clustering big data streams: Recent challenges and contributions". In: *IT - Information Technology* 58.4 (2016). Cited by: 6; All Open Access, Green Open Access, pp. 206–213. DOI: 10.1515/itit-2016-0007.

[Hen+16] C. Hennig, M. Meila, F. Murtagh, and R. Rocci. *Handbook of Cluster Analysis*. Chapman and Hall/CRC, 2016. DOI: 10.1201/b19706.

[LS16] Marco Locatelli and Fabio Schoen. "Global optimization based on local searches". In: *Annals of Operations Research* 240.1 (2016), pp. 251–270. DOI: 10.1007/s10479-015-2014-2.

[MTU16] Nenad Mladenovic, Raca Todosijevic, and Dragan Urosevic. "Less is more: Basic variable neighborhood search for minimum differential dispersion problem". In: *Information Sciences* (2016). DOI: 10.1016/j.ins.2015.07.044.

[Moh+16] Amin Mohebi, Saeed Aghabozorgi, Teh Ying Wah, Tutut Herawan, and Ramin Yahyapour. "Iterative big data clustering algorithms: a review". In: *Software-Practice & Experience* (2016). DOI: 10.1002/spe.2341.

[SKT16] Yousef Sharafi, Mojtaba Ahmadieh Khanesar, and Mohammad Teshnehlab. "COOA: Competitive optimization algorithm". In: *Swarm and Evolutionary Computation* 30 (2016), pp. 39–63. ISSN: 2210-6502. DOI: 10.1016/j.swevo.2016.04.002.

[SA16] Nazmul Siddique and Hojjat Adeli. "Simulated Annealing, Its Variants and Engineering Applications". In: *International Journal on Artificial Intelligence Tools* 25.6 (2016). DOI: 10.1142/S0218213016300015.

[SJ16] Ankita Sinha and Prasanta K. Jana. *Clustering algorithms for big data: A survey*. Cited by: 2. 2016, pp. 143–162. DOI: 10.1201/9781315368061.

[DGG15] Andrea De Mauro, Marco Greco, and Michele Grimaldi. "What is big data? A consensual definition and a review of key research topics". In: *AIP Conference Proceedings* 1644.1 (Feb. 2015), pp. 97–104. ISSN: 0094-243X. DOI: 10.1063/1.4907823.

[Des+15] S. Desale, A. Rasool, S. Andhale, and P. Rane. "Heuristic and Meta-Heuristic Algorithms and Their Relevance to the Real World: A Survey". In: *International Journal of Computer Engineering in Research Trends* (2015).

[He+15] Qing He, Haocheng Wang, Fuzhen Zhuang, Tianfeng Shang, and Zhongzhi Shi. "Parallel sampling from big data with uncertainty distribution". In: *Fuzzy Sets and Systems* 258 (2015). Special issue: Uncertainty in Learning from Big Data, pp. 117–133. ISSN: 0165-0114. DOI: 10.1016/j.fss.2014.01.016.

[HYW+15] C. Hong, J. Yu, J. Wan, et al. "Multimodal Deep Autoencoder for Human Pose Recovery". In: *IEEE Transactions on Image Processing* (2015). DOI: 10.1109/TIP.2015.2487860.

[Kar15] Napsu Karmitsa. “Diagonal Bundle Method for Nonsmooth Sparse Optimization”. In: *Journal of Optimization Theory and Applications* (2015). DOI: 10.1007/s10957-014-0666-8.

[LPS15] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. “Numba: a LLVM-based Python JIT compiler”. In: *LLVM ’15: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. Nov. 2015, pp. 1–6. DOI: 10.1145/2833157.2833162.

[OB15] Burak Ordin and Adil M. Bagirov. “A heuristic algorithm for solving the minimum sum-of-squares clustering problems”. In: *Journal of Global Optimization* (2015). DOI: 10.1007/s10898-014-0171-5.

[WKK15] Roland Winkler, Frank Klawonn, and Rudolf Kruse. “Prototype based fuzzy clustering algorithms in high-dimensional feature spaces”. In: *Studies in Fuzziness and Soft Computing* 322 (2015). Cited by: 1, pp. 233–243. DOI: 10.1007/978-3-319-16235-5_18.

[Yin+15] Yihang Yin, Fengzheng Liu, Xiang Zhou, and Quanzhong Li. “An Efficient Data Compression Model Based on Spatial Clustering and Principal Component Analysis in Wireless Sensor Networks”. In: *SENSORS* 15.8 (Aug. 2015), pp. 19443–19465. DOI: 10.3390/s150819443.

[Alt+14] Douglas S. Altner, Ravindra K. Ahuja, Ozlem Ergun, and James B. Orlin. *Very Large-Scale neighborhood search*. 2014, pp. 339–368. DOI: 10.1007/978-1-4614-6940-7_13.

[BS14] L.Yu. Barash and L.N. Shchur. “PRAND: GPU accelerated parallel random number generation library: Using most reliable algorithms and applying parallelism of modern GPUs and CPUs”. In: *Computer Physics Communications* 185.4 (2014), pp. 1343–1353. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2014.01.007.

[Cui+14] Huimin Cui, Gong Ruan, Jingling Xue, Rui Xie, Lei Wang, and Xiaobing Feng. "A collaborative divide-and-conquer K-means clustering algorithm for processing large data". In: *Proceedings of the 11th ACM Conference on Computing Frontiers* (2014). DOI: 10.1145/2597917.2597918.

[FAT+14] Adil Fahad, Najlaa Alshatri, Zahir Tari, et al. "A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis". In: *IEEE Transactions on Emerging Topics in Computing* (2014). DOI: 10.1109/TETC.2014.2330519.

[HM14] Pierre Hansen and Nenad Mladenovic. "Variable Neighborhood Search". In: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (2014).

[KH14] Jens Krüger and Markus Hadwiger. "Scalable devices". In: *Mathematics and Visualization* 37 (2014), pp. 361–373. DOI: 10.1007/978-1-4471-6497-5_29.

[A+13] Silva J. A., Faria E. R., Barros R. C., Hruschka E. R., and Carvalho A. "Data stream clustering: A survey". In: *ACM Computing Surveys* (2013). DOI: 10.1145/2522968.2522981.

[Cor13] Jyrko Correa-Morris. "An indication of unification for different clustering approaches". In: *Pattern Recognition* 46.9 (2013), pp. 2548–2561. DOI: 10.1016/j.patcog.2013.02.016.

[Par+13] Eduardo G. Pardo, Nenad Mladenović, Juan J. Pantrigo, and Abraham Duarte. "Variable Formulation Search for the Cutwidth Minimization Problem". In: *Applied Soft Computing* 13.5 (2013), pp. 2242–2252. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2013.01.016.

[RRR13] Dick de Ridder, Jeroen de Ridder, and Marcel J. T. Reinders. "Pattern recognition in bioinformatics". In: *BRIEFINGS IN BIOINFORMATICS* 14.5 (Sept. 2013), pp. 633–647. ISSN: 1467-5463. DOI: 10.1093/bib/bbt020.

[SSE13] Amit Sabne, Putt Sakdhnagool, and Rudolf Eigenmann. "Scaling Large-Data Computations on Multi-GPU Accelerators". In: *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*. ICS '13. Eugene, Oregon, USA: Association for Computing Machinery, 2013, pp. 443–454. ISBN: 9781450321303. DOI: 10.1145/2464996.2465023.

[SS13] Seref Sagiroglu and Duygu Sinanc. "Big data: A review". In: *2013 International Conference on Collaboration Technologies and Systems (CTS)*. 2013, pp. 42–47. DOI: 10.1109/CTS.2013.6567202.

[B+12] Bahmani B., Moseley B., Vattani A., Kumar R., and Vassilvitskii S. "Scalable k-means++". In: *Proceedings of the VLDB Endowment* (2012). DOI: 10.14778/2180912.2180915.

[MC12] Fionn Murtagh and Pedro Contreras. "Algorithms for hierarchical clustering: An overview". In: *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery* 2 (Jan. 2012), pp. 86–97. DOI: 10.1002/widm.53.

[Rak+12] Thanawin Rakthanmanon, Eamonn J. Keogh, Stefano Lonardi, and Scott Evans. "MDL-based time series clustering". In: *KNOWLEDGE AND INFORMATION SYSTEMS* 33.2 (Nov. 2012), pp. 371–399. ISSN: 0219-1377. DOI: 10.1007/s10115-012-0508-7.

[YLL12] Miin-Shen Yang, Chien-Yo Lai, and Chih-Ying Lin. "A robust EM clustering algorithm for Gaussian mixture models". In: *Pattern Recognition* 45.11 (2012), pp. 3950–3961. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2012.04.031.

[Vat11] Andrea Vattani. "K-means requires exponentially many iterations even in the plane". In: *Discrete Computational Geometry* (2011). DOI: 10.1007/s00454-011-9340-1.

[WS11] David Williamson and David Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. ISBN: 9780521195270.

[ZZ11] Peixin Zhao and Cun-Quan Zhang. "A new clustering method and its application in social networks". In: *PATTERN RECOGNITION LETTERS* 32.15 (Nov. 2011), pp. 2109–2118. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2011.06.008.

[BUW10] Adil M. Bagirov, Julien Ugon, and Dean Webb. "Fast modified global k-means algorithm for incremental cluster construction". In: *Pattern Recognition* (2010). DOI: 10.1016/j.patcog.2010.10.018.

[BB10] Roberto Battiti and Mauro Brunato. "Reactive Search Optimization: Learning While Optimizing". In: *Mathematical Programming* (2010). DOI: 10.1007/978-1-4419-1665-5_18.

[Jai10] A. K. Jain. "Data clustering: 50 years beyond K-means". In: *Pattern Recognition Letters* (2010). DOI: 10.1016/j.patrec.2009.09.011.

[LOT10] H.R. Lourenco, Martin O., and Stutzle T. *Iterated Local Search: Framework and Applications. Handbook of Metaheuristics*. Kluwer Academic Publishers, 2010. DOI: 10.1007/978-1-4419-1665-5_12.

[Nin+10] Hao Ning, Wei Xu, Yihong Chi, Yihong Gong, and Thomas S Huang. "Incremental spectral clustering by efficiently updating the eigen-system". In: *Pattern Recognition* 43.1 (2010), pp. 113–127.

[Scu10] David Sculley. "Web-scale k-means clustering". In: *Proceedings of the 19th international conference on World wide web* (2010), pp. 1177–1178.

[Alo+09] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. "NP-hardness of Euclidean sum-of-squares clustering". In: *Machine Learning* (2009). DOI: 10.1007/s10994-009-5103-0.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[OA09] Tansel Ozyer and Reda Alhajj. "Parallel clustering of high dimensional data by integrating multi-objective genetic algorithm with divide and conquer". In: *Applied Intelligence* (2009). DOI: 10.1007/s10489-008-0129-8.

[ZMH09] W. Zhao, H. Ma, and Q. He. "Parallel k-means clustering based on MapReduce". In: *Cloud Computing*. Vol. 5931. Springer, 2009, pp. 674–679.

[Bag08] Adil M. Bagirov. "Modified global k-means algorithm for minimum sum-of-squares clustering problems". In: *Pattern Recognition* (2008). DOI: 10.1016/j.patcog.2008.04.004.

[Chh+08] Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, William Macy, Mostafa Hagog, Yen-Kuang Chen, Akram Baransi, Sanjeev Kumar, and Pradeep Dubey. "Efficient Implementation of Sorting on Multi-Core SIMD CPU Architecture". In: *Proc. VLDB Endow.* 1.2 (Aug. 2008), pp. 1313–1324. ISSN: 2150-8097. DOI: 10.14778/1454159.1454171.

[DG08] J. Dean and S. Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM*. Vol. 51. 1. 2008, pp. 107–113.

[DWV08] Benoit Depaire, Geert Wets, and Koen Vanhoof. "Traffic accident segmentation by means of latent class clustering". In: *ACCIDENT ANALYSIS AND PREVENTION* 40.4 (July 2008), pp. 1257–1266. ISSN: 0001-4575. DOI: 10.1016/j.aap.2008.01.007.

[Fil+08] Maurizio Filippone, Francesco Camastra, Francesco Masulli, and Stefano Rovetta. "A survey of kernel and spectral methods for clustering". In: *Pattern Recognition* (2008). DOI: 10.1016/j.patcog.2007.05.018.

[HPZ08] Alain Hertz, Matthieu Plumettaz, and Nicolas Zufferey. "Variable space search for graph coloring". In: *DISCRETE APPLIED MATHEMATICS* 156.13 (July 2008). 5th International Conference on Graphs and Optimization, Leukerbad,

SWITZERLAND, AUG, 2006, pp. 2551–2560. DOI: 10.1016/j.dam.2008.03.022.

[AV07] David Arthur and Sergei Vassilvitskii. "How much can k-means be improved by using better initialization and repeats?" In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (2007).

[AL07] Esmaeil Atashpaz-Gargari and Caro Lucas. "Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition". In: *2007 IEEE Congress on Evolutionary Computation*. 2007, pp. 4661–4667. DOI: 10.1109/CEC.2007.4425083.

[HMM07] N. Haarala, K. Miettinen, and M. Makela. "Globally convergent limited memory bundle method for large-scale nonsmooth optimization". In: *Mathematical Programming* (2007). DOI: 10.1007/s10107-006-0728-2.

[Lux07] Ulrike von Luxburg. "A tutorial on spectral clustering". English. In: *STATISTICS AND COMPUTING* 17.4 (Dec. 2007), pp. 395–416. ISSN: 0960-3174. DOI: 10.1007/s11222-007-9033-z.

[MPU07] Nenad Mladenović, Frank Plastria, and Dragan Urošević. "Formulation Space Search for Circle Packing Problems". In: *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*. Ed. by Thomas Stützle, Mauro Birattari, and Holger H. Hoos. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 212–216. ISBN: 978-3-540-74446-7.

[SSP07] Vijay A. Saraswat, Vivek Sarkar, and Christoph von Praun. "X10: Concurrent Programming for Modern Architectures". In: *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP '07. San Jose, California, USA: Association for Computing Machinery, 2007, p. 271. ISBN: 9781595936028. DOI: 10.1145/1229428.1229483.

[WKQ+07] Xindong Wu, Vipin Kumar, J. Ross Quinlan, et al. "Top 10 algorithms in data mining". In: *Knowledge and Information Systems* (2007). DOI: 10.1007/s10115-007-0114-2.

[Teh+06] Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. "Hierarchical Dirichlet Processes". In: *Journal of the American Statistical Association* 101.476 (2006), pp. 1566–1581. ISSN: 01621459.

[HS05] Holger H. Hoos and Thomas Stützle. "Stochastic Local Search: Foundations and Applications". In: *Stochastic Local Search*. Ed. by Holger H. Hoos and Thomas Stützle. The Morgan Kaufmann Series in Artificial Intelligence. San Francisco: Morgan Kaufmann, 2005, pp. 113–147. ISBN: 978-1-55860-872-6. DOI: 10.1016/B978-155860872-6/50020-2.

[XW05] Rui Xu and D. Wunsch. "Survey of clustering algorithms". In: *IEEE Transactions on Neural Networks* (2005). DOI: 10.1109/TNN.2005.845141.

[JTZ04] DX Jiang, C Tang, and AD Zhang. "Cluster analysis for gene expression data: A survey". In: *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING* 16.11 (Nov. 2004), pp. 1370–1386. ISSN: 1041-4347. DOI: 10.1109/TKDE.2004.68.

[Agg+03] Charu C. Aggarwal, Philip S. Yu, Jiawei Han, and Jianyong Wang. "- A Framework for Clustering Evolving Data Streams". In: *Proceedings 2003 VLDB Conference*. Ed. by Johann-Christoph Freytag, Peter Lockemann, Serge Abiteboul, Michael Carey, Patricia Selinger, and Andreas Heuer. San Francisco: Morgan Kaufmann, 2003, pp. 81–92. ISBN: 978-0-12-722442-8. DOI: 10.1016/B978-012722442-8/50016-1.

[Elk03] Charles Elkan. "Using the triangle inequality to accelerate k-means". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. Washington, DC, USA: AAAI Press, 2003, pp. 147–153. ISBN: 1577351894.

[HM03] P. Hansen and N. Mladenovic. "Variable neighborhood search". In: *F. Glover and G. Kochenberger (eds.), Handbook of Metaheuristics Kluwer* (2003).

[HK03] A. Hinneburg and D. Keim. *A General Approach to Clustering in Large Databases with Noise*. 5. Knowledge and Information Systems, 2003, pp. 387–415. DOI: 10.1007/s10115-003-0086-9.

[LVV03] Aristidis Likas, Nikos Vlassis, and Jakob J. Verbeek. "The global k-means clustering algorithm". In: *Pattern Recognition* (2003). DOI: 10.1016/S0031-3203(02)00060-2.

[AHK01] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. "On the Surprising Behavior of Distance Metrics in High Dimensional Space". In: *Lecture Notes in Computer Science* (2001). DOI: 10.1007/3-540-44503-X_27.

[HM01] Pierre Hansen and Nenad Mladenovic. "J-Means: A new local search heuristic for minimum sum of squares clustering". In: *Pattern Recognition* (2001). DOI: 10.1016/S0031-3203(99)00216-2.

[May01] John M. May. *Parallel I/O for High Performance Computing*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001. ISBN: 1558606645.

[CW00] Isom L. Crawford and Kevin R. Wadleigh. *Software Optimization for High Performance Computers*. USA: Prentice Hall PTR, 2000. ISBN: 0130170089.

[MNU00] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. "Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching". In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '00. Boston, Massachusetts, USA: Association for Computing Machinery, 2000, pp. 169–178. ISBN: 1581132336. DOI: 10.1145/347090.347123.

[Ank+99] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. "OPTICS: Ordering Points to Identify the Clustering Structure". In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. SIGMOD '99. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1999, pp. 49–60. ISBN: 1581130848. DOI: 10.1145/304182.304187.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. "Data clustering: a review". In: *ACM Computing Surveys* (1999). DOI: 10.1145/331499.331504.

[Pen99] J.M Pena. "An empirical comparison of four initialization methods for the K-means algorithm". In: *Pattern Recognition Letters* (1999). DOI: 10.1016/S0167-8655(99)00069-0.

[Agr+98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications". In: *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*. SIGMOD '98. Seattle, Washington, USA: Association for Computing Machinery, 1998, pp. 94–105. ISBN: 0897919955. DOI: 10.1145/276304.276314.

[GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. "CURE: An Efficient Clustering Algorithm for Large Databases". In: *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*. SIGMOD '98. Seattle, Washington, USA: Association for Computing Machinery, 1998, pp. 73–84. ISBN: 0897919955. DOI: 10.1145/276304.276312.

[MN98] Andrew McCallum and Kamal Nigam. "A Comparison of Event Models for Naive Bayes Text Classification". In: *Proceedings of the AAAI/ICML-98 Workshop on Learning for Text Categorization*. Madison, Wisconsin: AAAI Press, 1998, pp. 41–48.

[YYL98] M Yeung, BL Yeo, and B Liu. "Segmentation of video by clustering and graph analysis". In: *COMPUTER VISION AND IMAGE UNDERSTANDING* 71.1 (July 1998), pp. 94–109. ISSN: 1077-3142. DOI: 10.1006/cviu.1997.0628.

[HJ97] P. Hansen and B. Jaumard. "Cluster analysis and mathematical programming". In: *Mathematical Programming* (1997). DOI: 10.1007/BF02614317.

[MH97] N. Mladenovic and P. Hansen. "Variable neighborhood search". In: *Computers and Operations Research* (1997). DOI: 10.1016/S0305-0548(97)00031-2.

[WYM97] Wei Wang, Jiong Yang, and Richard R. Muntz. "STING: A Statistical Information Grid Approach to Spatial Data Mining". In: *Proceedings of the 23rd International Conference on Very Large Data Bases*. VLDB '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 186–195. ISBN: 1558604707.

[Est+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei 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: AAAI Press, 1996, pp. 226–231.

[GSS96] M. Gupta, E. Schonberg, and H. Srinivasan. "A unified framework for optimizing communication in data-parallel programs". In: *IEEE Transactions on Parallel and Distributed Systems* 7.7 (1996), pp. 689–704. DOI: 10.1109/71.508249.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. "BIRCH: an efficient data clustering method for very large databases". In: *ACM SIGMOD Record* (1996). DOI: 10.1145/235968.233324.

[KR90a] Leonard Kaufman and Peter J. Rousseeuw. "Clustering Large Applications (Program CLARA)". In: *Finding Groups in Data*. John Wiley & Sons, Ltd,

1990. Chap. 3, pp. 126–163. ISBN: 9780470316801. DOI: 10.1002/9780470316801.ch3.

[KR90b] Leonard Kaufman and Peter J. Rousseeuw. "Partitioning Around Medoids (Program PAM)". In: *Finding Groups in Data*. John Wiley & Sons, Ltd, 1990. Chap. 2, pp. 68–125. ISBN: 9780470316801. DOI: 10.1002/9780470316801.ch2.

[JD88] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Vol. 7. Prentice Hall, 1988.

[Smi87] Douglas R. Smith. "Applications of a strategy for designing divide-and-conquer algorithms". In: *Science of Computer Programming* (1987). DOI: 10.1016/0167-6423(87)90034-7.

[HJ82] P. Hansen and B. Jaumard. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* (1982). DOI: 10.1109/TIT.1982.1056489.

[HW79] J. A. Hartigan and M. A. Wong. "Algorithm AS 136: A K-Means Clustering Algorithm". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* (1979). DOI: 10.2307/2346830.

[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 (1977), pp. 1–22. DOI: 10.1111/j.2517-6161.1977.tb01600.x.

[Mac67] J. MacQueen. "Some methods for classification and analysis of multivariate observations". In: *5-th Berkeley Symposium on Mathematical Statistics and Probability* (1967).

[How66] R. Howard. "Classifying a population into homogeneous groups". In: *Operational Research in the Social Sciences* (1966).

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

[Jr63] Joe H. Ward Jr. “Hierarchical Grouping to Optimize an Objective Function”. In: *Journal of the American Statistical Association* (1963).

Appendix A

EXPERIMENTS WITH BIG-MEANS

Clustering details include the parameters and the following attributes of the clustering process:

- k is the total number of clusters;
- f^* is the best known objective function (1.1) value multiplied by the number provided after the name of the dataset in the caption of each table;
- s is the sample size;
- n_{exec} is the number of executions for each choice of k;
- n_s is the number of used samples;
- n_{full} is the number of iterations for the full dataset clustering;
- t_{init} is the CPU time used for the initialization phase;
- t_{full} is the CPU time used for the full dataset clustering after initialization;
- T is the maximal CPU time allowed for the execution of an algorithm;
- n_d is the number of distance function evaluations.

A.1 CORD-19 Embeddings

Table A.1: Summary of the results with CORD-19 Embeddings ($\times 10^9, m = 599616, n = 768$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	2.03893*	0.0	0.01	0.01	3.88	19.24	34.67	0.0	0.0	0.0	1.7	2.82	7.05	–	–	0.0	0.0	0.0	31.15	34.18	36.79	5.58	18.92	27.7	9.3	14.68	19.51	–	–
3	1.9093*	0.01	0.59	4.03	5.5	20.58	33.62	0.0	0.01	0.01	4.92	8.96	18.49	–	–	0.0	0.01	0.02	49.51	52.39	55.02	13.78	21.09	29.73	15.49	19.73	27.69	–	–
5	1.77676*	0.01	0.09	0.15	14.54	29.36	39.94	0.0	0.13	0.7	7.56	13.46	19.69	–	–	0.0	0.28	0.97	74.39	83.05	87.47	12.09	17.73	24.5	24.11	31.24	39.07	–	–
10	1.62555*	0.1	0.36	0.71	9.76	26.98	39.11	0.01	0.52	0.84	15.73	29.74	51.19	–	–	0.14	0.38	0.72	163.66	180.43	194.84	16.28	18.28	21.36	53.86	61.09	68.28	–	–
15	1.55295*	-0.08	0.44	1.22	14.67	25.23	32.25	0.01	0.18	0.42	41.28	66.01	90.44	–	–	-0.01	0.28	0.88	261.41	270.81	294.0	16.52	18.38	21.08	80.38	89.89	98.41	–	–
20	1.49987*	0.2	0.46	0.75	20.22	31.07	40.63	0.11	0.46	1.14	47.68	76.92	101.11	–	–	-0.03	0.38	0.95	329.03	361.7	382.1	16.95	18.92	20.41	101.27	116.06	136.17	–	–
25	1.46394*	-0.01	0.16	0.32	23.2	30.59	39.53	0.07	0.54	1.11	72.88	98.6	153.11	–	–	-0.02	0.36	0.85	424.22	460.43	501.76	17.18	19.78	23.55	127.14	143.11	159.76	–	–
Mean:		**0.3**			**26.15**			**0.26**			**42.36**			–	–	**0.24**			**206.14**			**19.01**			**67.97**			–	–

Table A.2: Clustering details with CORD-19 Embeddings

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	7	32000	82	40.0	19.24	1.5E+07	5	6.3E+06	–	–	–	–	31.0	3.18	6	9.1E+06	14.68	0.0	3	3.4E+06	–	–
3	7	32000	71	40.0	20.58	2.1E+07	13	2.4E+07	–	–	–	–	44.98	7.42	10	2.2E+07	19.73	0.0	4	7.2E+06	–	–
5	7	32000	84	40.0	29.36	4.2E+07	14	4.1E+07	–	–	–	–	70.05	13.0	12	4.3E+07	31.23	0.01	7	2.0E+07	–	–
10	7	32000	39	40.0	26.98	5.0E+07	18	1.1E+08	–	–	–	–	150.89	29.54	17	1.2E+08	61.08	0.01	5	2.8E+07	–	–
15	7	32000	18	40.0	25.23	4.3E+07	27	2.4E+08	–	–	–	–	216.18	54.64	23	2.3E+08	89.87	0.01	7	6.3E+07	–	–
20	7	32000	12	40.0	31.07	5.5E+07	25	3.0E+08	–	–	–	–	286.61	75.08	24	3.2E+08	116.04	0.02	6	7.7E+07	–	–
25	7	32000	7	40.0	30.59	4.8E+07	26	3.9E+08	–	–	–	–	367.74	92.69	25	4.1E+08	143.08	0.03	7	1.1E+08	–	–

A.2 HEPMASS

Table A.3: Summary of the results with HEPMASS ($\times 10^8, m = 10500000, n = 27$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	2.48889*	0.0	0.0	0.0	5.86	15.32	25.73	0.0	0.0	0.0	4.85	5.69	6.3	–	–	0.0	0.0	0.0	16.03	17.97	19.11	7.46	12.74	16.65	6.54	8.76	11.29	–	–
3	2.36789*	0.0	0.36	1.26	7.27	21.98	29.86	0.0	0.78	1.58	6.85	10.48	14.32	–	–	0.0	0.19	1.28	24.17	28.18	33.4	9.94	12.98	16.12	9.48	13.81	17.47	–	–
5	2.21106*	0.33	0.46	0.77	1.13	6.75	13.03	0.01	0.49	1.14	10.51	15.62	20.06	–	–	-0.0	0.31	0.82	37.33	44.41	53.04	10.44	12.64	15.67	15.76	18.54	21.25	–	–
10	2.00353*	0.12	0.43	0.93	8.88	23.22	28.9	0.33	0.8	1.31	18.01	29.18	44.41	–	–	0.1	0.47	0.78	76.59	86.06	94.89	10.41	13.23	16.91	28.04	33.34	39.77	–	–
15	1.89922*	0.09	0.44	0.79	3.21	16.11	28.36	0.15	0.31	0.55	28.63	45.52	66.15	–	–	0.04	0.53	0.95	122.5	126.62	134.37	11.51	12.85	14.81	46.65	49.36	51.01	–	–
20	1.82904*	0.17	0.35	0.51	7.28	17.44	28.86	0.19	0.49	0.85	27.6	53.06	78.02	–	–	0.21	0.38	0.65	158.12	180.97	197.43	12.25	14.43	17.03	58.2	61.38	64.01	–	–
25	1.77524*	0.12	0.38	0.66	4.57	15.09	28.27	0.04	0.36	0.53	48.82	72.82	127.82	–	–	0.19	0.52	0.91	200.27	220.95	240.8	12.02	13.29	16.65	65.68	75.23	83.54	–	–
Mean:		**0.35**			**16.56**			**0.46**			**33.2**			–	–	**0.34**			**100.74**			**13.16**			**37.2**			–	–

Table A.4: Clustering details with HEPMASS

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	7	64000	391	30.0	15.32	1.0E+08	8	1.6E+08	–	–	–	–	13.41	4.57	7	1.9E+08	8.76	0.0	3	6.0E+07	–	–
3	7	64000	492	30.0	21.98	2.5E+08	14	4.5E+08	–	–	–	–	18.73	9.46	13	4.9E+08	13.81	0.0	4	1.3E+08	–	–
5	7	64000	126	30.0	6.75	1.3E+08	20	1.0E+09	–	–	–	–	30.33	14.08	18	1.1E+09	18.54	0.0	5	2.7E+08	–	–
10	7	64000	382	30.0	23.22	7.6E+08	27	2.8E+09	–	–	–	–	62.37	23.69	22	2.6E+09	33.34	0.0	7	7.1E+08	–	–
15	7	64000	157	30.0	16.11	5.5E+08	31	4.8E+09	–	–	–	–	91.4	35.22	24	4.2E+09	49.36	0.0	6	1.0E+09	–	–
20	7	64000	148	30.0	17.44	8.1E+08	28	5.8E+09	–	–	–	–	122.4	58.57	31	7.2E+09	61.38	0.0	8	1.8E+09	–	–
25	7	64000	104	30.0	15.09	8.2E+08	33	8.6E+09	–	–	–	–	150.32	70.62	31	9.0E+09	75.23	0.0	8	2.1E+09	–	–

A.3 US Census Data 1990

Table A.5: Summary of the results with US Census Data 1990 ($\times 10^8, m = 2458285, n = 68$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	18.39812*	0.06	0.32	0.92	0.17	1.73	2.98	0.0	0.0	0.0	0.41	0.73	1.16	–	–	0.0	0.0	0.0	7.99	9.0	9.7	0.67	1.51	3.68	3.42	5.1	7.28	0.0	677.28
3	6.1591*	0.04	24.69	164.26	0.12	1.62	2.94	0.0	114.5	198.71	0.72	1.33	3.32	–	–	0.0	8.62	172.43	12.0	14.02	17.12	2.1	163.9	201.34	4.6	7.12	10.08	0.0	724.9
5	3.35214*	0.08	5.57	28.24	0.12	1.85	2.98	0.0	230.71	448.85	0.86	2.31	4.09	–	–	0.0	3.18	24.24	20.81	23.27	26.81	4.61	241.44	401.54	8.49	11.18	14.76	10.29	838.16
10	2.36352*	1.46	6.06	11.98	0.08	1.88	2.96	0.0	90.31	544.33	3.14	6.35	12.75	–	–	0.01	6.36	15.47	45.96	49.19	56.94	11.07	102.98	541.05	17.92	21.89	24.82	4.12	1622.21
15	2.04097*	2.08	5.44	10.26	0.23	1.65	3.0	2.66	39.3	623.7	5.96	11.67	25.17	–	–	0.47	5.26	9.94	69.9	77.83	86.34	13.51	84.32	632.84	26.66	30.62	36.26	2.74	2570.91
20	1.81278*	2.08	4.91	9.03	0.24	1.78	2.82	3.44	46.28	701.04	8.29	18.59	32.65	–	–	0.9	4.78	10.4	91.74	101.52	115.32	16.61	26.09	35.43	33.52	39.79	45.81	4.64	3299.06
25	1.64602*	2.53	4.97	8.17	0.29	1.51	3.04	3.77	14.53	32.93	11.39	23.37	39.38	–	–	1.39	5.56	11.88	112.02	126.33	144.66	17.9	25.38	39.04	42.69	49.02	57.14	7.87	4432.8
Mean:			**7.42**			**1.72**			**76.52**			**9.19**		–	–		**4.82**			**57.31**			**92.23**			**23.53**		**4.24**	**2023.62**

Table A.6: Clustering details with US Census Data 1990

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	20	6000	241	3.0	1.73	5.8E+06	3	1.6E+07	–	–	–	–	8.41	0.58	2	2.0E+07	5.1	0.0	3	1.4E+07	677.28	-2.1E+09
3	20	6000	208	3.0	1.62	7.6E+06	5	3.7E+07	–	–	–	–	13.39	0.63	2	3.3E+07	7.12	0.0	3	2.3E+07	724.9	-2.1E+09
5	20	6000	190	3.0	1.85	1.4E+07	7	8.2E+07	–	–	–	–	21.15	2.12	6	1.1E+08	11.18	0.0	5	6.5E+07	838.16	-2.1E+09
10	20	6000	122	3.0	1.88	2.6E+07	11	2.7E+08	–	–	–	–	42.92	6.27	11	3.3E+08	21.88	0.0	5	1.3E+08	1622.21	-2.1E+09
15	20	6000	74	3.0	1.65	2.9E+07	14	5.1E+08	–	–	–	–	66.57	11.26	13	6.0E+08	30.61	0.01	6	2.1E+08	2570.91	-2.1E+09
20	20	6000	53	3.0	1.78	3.4E+07	17	8.4E+08	–	–	–	–	87.03	14.49	13	8.0E+08	39.78	0.0	6	3.0E+08	3299.06	-2.1E+09
25	20	6000	32	3.0	1.51	3.2E+07	18	1.1E+09	–	–	–	–	108.29	18.04	14	1.0E+09	49.02	0.01	7	4.1E+08	4432.8	-2.1E+09

A.4 Gisette

Table A.7: Summary of the results with Gisette ($\times 10^{12}, m = 13500, n = 5000$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	4.19944	0.01	0.01	0.01	7.53	29.13	58.69	0.0	0.01	0.01	0.42	0.61	0.82	–	–	0.0	0.01	0.02	3.17	3.43	3.88	10.05	13.51	17.87	1.73	2.14	3.34	0.0	112.61
3	4.11596	0.01	0.05	0.35	4.2	32.18	58.53	0.01	0.05	0.07	0.79	1.28	2.13	–	–	0.01	0.03	0.07	4.84	5.37	6.05	10.04	13.25	17.61	1.87	2.8	3.74	0.0	209.76
5	4.02303	0.02	0.08	0.26	9.94	34.02	59.35	0.02	0.11	0.17	1.28	1.8	2.63	–	–	0.03	0.11	0.25	8.03	8.9	10.22	11.08	13.57	17.3	3.39	4.83	5.87	0.0	466.11
10	3.87672	0.05	0.14	0.27	14.2	34.52	55.71	0.06	0.19	0.32	2.73	4.61	7.49	–	–	0.06	0.13	0.25	16.42	18.85	21.15	11.28	12.69	14.7	6.86	8.76	10.83	0.02	1345.13
15	3.81766	-0.44	-0.3	-0.16	17.59	41.05	58.45	-0.4	-0.31	-0.16	4.37	5.93	9.1	–	–	-0.38	-0.28	-0.14	24.33	27.01	28.91	10.59	12.22	13.6	10.62	12.71	14.57	0.27	2176.84
20	3.81436	-1.75	-1.66	-1.53	28.08	48.48	58.72	-1.74	-1.66	-1.57	5.0	7.51	11.21	–	–	-1.76	-1.66	-1.49	33.22	36.65	40.5	9.6	10.52	11.55	14.17	16.37	19.05	-1.36	3495.14
25	3.74937	-1.2	-1.09	-0.94	31.42	47.6	61.52	-1.18	-1.09	-0.97	8.19	10.54	14.97	–	–	-1.23	-1.09	-0.98	42.22	43.85	45.47	10.25	11.22	12.34	18.01	20.33	24.18	0.22	3996.96
Mean:			**-0.4**			**38.14**			**-0.39**			**4.61**		–	–		**-0.39**			**20.58**			**12.43**			**9.71**		**-0.12**	**1686.08**

Table A.8: Clustering details with Gisette

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	15	10000	72	60.0	29.13	4.3E+06	9	2.4E+05	–	–	–	–	2.79	0.64	9	3.0E+05	2.14	0.0	2	5.8E+04	112.61	1.4E+07
3	15	10000	65	60.0	32.18	6.3E+06	14	5.7E+05	–	–	–	–	4.28	1.1	12	5.9E+05	2.8	0.0	3	1.2E+05	209.76	2.6E+07
5	15	10000	51	60.0	34.02	8.5E+06	13	9.0E+05	–	–	–	–	7.0	1.9	14	1.1E+06	4.82	0.01	4	2.6E+05	466.11	5.9E+07
10	15	10000	27	60.0	34.52	1.0E+07	19	2.5E+06	–	–	–	–	14.09	4.76	20	3.1E+06	8.75	0.02	4	6.0E+05	1345.13	1.7E+08
15	15	10000	20	60.0	41.05	1.3E+07	18	3.6E+06	–	–	–	–	21.25	5.76	18	4.2E+06	12.67	0.05	6	1.2E+06	2176.84	2.8E+08
20	15	10000	17	60.0	48.48	1.6E+07	19	5.0E+06	–	–	–	–	28.07	8.58	21	6.5E+06	16.31	0.06	6	1.6E+06	3495.14	4.5E+08
25	15	10000	8	60.0	47.6	1.4E+07	20	6.9E+06	–	–	–	–	35.08	8.77	18	7.0E+06	20.24	0.1	6	2.0E+06	3996.96	5.3E+08

A.5 Music Analysis

Table A.9: Summary of the results with Music Analysis ($\times 10^{11}, m = 106574, n = 518$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	5.00474*	0.01	2.7	26.32	0.43	3.82	7.56	0.0	2.6	26.0	0.32	0.56	0.76	–	–	0.0	2.6	26.0	2.22	2.47	2.66	1.19	7.63	33.65	1.44	1.97	2.86	-0.0	19.67
3	3.83748*	0.03	0.11	0.29	0.38	4.19	8.0	0.0	5.0	9.11	0.58	1.05	2.18	–	–	0.0	2.27	9.1	3.56	4.03	5.07	1.73	13.13	29.62	2.11	2.65	3.47	-0.0	41.94
5	2.74249*	0.14	2.09	5.39	1.02	4.44	7.9	0.01	1.23	5.15	0.92	1.87	2.76	–	–	0.01	1.72	5.14	5.58	6.42	8.42	2.58	13.31	33.26	3.16	4.08	5.49	-0.0	126.28
10	1.87296*	0.24	1.6	3.45	0.92	4.72	7.96	0.02	1.53	4.11	4.96	7.72	13.03	–	–	0.01	1.7	7.59	11.73	15.01	20.84	11.75	17.55	24.94	6.82	8.14	10.06	-0.02	312.92
15	1.54422*	0.58	1.42	2.55	1.18	4.92	8.02	0.15	0.96	2.23	6.26	12.94	24.7	–	–	0.05	0.61	1.78	20.21	25.92	32.51	10.75	19.46	28.73	9.25	11.47	13.25	0.51	684.25
20	1.35315*	0.72	1.52	4.35	1.5	4.9	8.11	0.14	1.0	2.59	10.82	20.23	39.86	–	–	-0.04	0.66	2.25	28.14	34.79	47.64	16.63	23.09	35.44	13.28	16.05	18.58	0.08	1087.6
25	1.22622*	0.98	2.22	5.51	1.73	5.62	8.45	0.12	1.2	2.59	17.82	28.18	51.96	–	–	0.09	0.58	1.77	38.43	46.17	59.58	14.14	24.16	37.81	17.35	19.55	22.36	1.64	1630.65
Mean:			**1.67**			**4.66**			**1.93**			**10.36**		–	–		**1.45**			**19.26**			**16.9**			**9.13**		**0.32**	**557.62**

Table A.10: Clustering details with Music Analysis

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	20	6000	157	8.0	3.82	6.6E+06	10	2.1E+06	–	–	–	–	1.95	0.51	9	2.4E+06	1.97	0.0	3	6.9E+05	19.67	1.8E+07
3	20	6000	114	8.0	4.19	9.5E+06	14	4.5E+06	–	–	–	–	3.01	1.02	13	5.0E+06	2.65	0.0	4	1.3E+06	41.94	4.2E+07
5	20	6000	73	8.0	4.44	1.2E+07	16	8.7E+06	–	–	–	–	4.78	1.64	15	9.3E+06	4.08	0.0	6	3.1E+06	126.28	1.4E+08
10	20	6000	27	8.0	4.72	1.8E+07	37	3.9E+07	–	–	–	–	10.05	4.96	24	2.9E+07	8.13	0.01	7	7.4E+06	312.92	3.6E+08
15	20	6000	15	8.0	4.92	1.9E+07	44	7.0E+07	–	–	–	–	15.32	10.6	35	6.1E+07	11.46	0.01	9	1.5E+07	684.25	8.0E+08
20	20	6000	9	8.0	4.9	1.9E+07	51	1.1E+08	–	–	–	–	19.17	15.62	40	9.1E+07	16.03	0.02	11	2.3E+07	1087.6	1.3E+09
25	20	6000	8	8.0	5.62	2.2E+07	58	1.5E+08	–	–	–	–	24.18	21.98	45	1.3E+08	19.52	0.03	11	3.0E+07	1630.65	2.0E+09

A.6 Protein Homology

Table A.11: Summary of the results with Protein Homology ($\times 10^{11}, m = 145751, n = 74$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	15.20433*	0.09	1.44	1.89	0.52	1.55	3.29	1.83	1.83	1.83	0.07	0.17	0.22	–	–	0.0	1.1	1.83	0.31	0.39	0.52	3.23	10.82	56.21	0.21	0.33	0.45	0.0	0.72
3	8.07129*	0.12	0.77	1.76	0.3	2.19	3.32	0.0	0.01	0.01	0.52	0.64	0.78	–	–	0.0	0.0	0.01	0.46	0.72	1.18	64.33	81.39	121.11	0.28	0.48	0.62	0.0	5.77
5	5.30537*	0.23	0.63	2.0	0.62	1.89	3.47	0.04	0.04	0.04	0.62	0.76	0.92	–	–	0.03	0.25	0.43	0.92	1.16	1.4	114.28	140.19	156.6	0.6	0.81	1.0	0.41	13.49
10	3.3767*	0.05	1.75	18.99	1.36	2.14	3.31	18.18	18.42	19.32	1.94	2.3	2.78	–	–	0.03	3.71	18.4	1.96	2.83	5.46	41.51	106.61	261.84	1.12	1.38	1.67	3.93	49.74
15	2.86473*	0.18	0.72	1.37	1.91	2.78	5.23	24.08	25.37	26.2	2.26	3.75	6.58	–	–	0.1	0.56	1.38	2.72	4.3	6.18	58.26	93.39	289.39	1.78	2.14	2.41	1.05	119.29
20	2.5732*	0.27	1.23	1.82	1.65	2.8	3.37	28.37	28.77	29.2	4.93	6.85	9.46	–	–	0.41	1.34	2.4	4.47	5.61	7.11	63.33	71.18	81.51	2.32	2.7	2.97	2.03	224.72
25	2.38539*	0.37	1.37	2.36	2.0	3.06	4.31	32.05	32.3	32.7	5.22	7.56	10.65	–	–	0.4	1.17	2.09	5.59	6.97	8.47	66.97	74.35	85.47	3.08	3.59	4.04	0.75	330.67
Mean:			**1.13**			**2.34**			**15.25**			**3.15**		–	–		**1.16**			**3.14**			**82.56**			**1.63**		**1.17**	**106.34**

Table A.12: Clustering details with Protein Homology

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	15	56000	41	3.5	1.55	1.3E+07	10	3.0E+06	–	–	–	–	0.29	0.1	6	2.4E+06	0.33	0.0	5	1.4E+06	0.72	2.0E+06
3	15	56000	50	3.5	2.19	2.8E+07	35	1.5E+07	–	–	–	–	0.44	0.28	15	7.5E+06	0.48	0.0	6	2.8E+06	5.77	2.8E+07
5	15	56000	30	3.5	1.89	3.2E+07	34	2.4E+07	–	–	–	–	0.76	0.4	16	1.4E+07	0.81	0.0	10	7.6E+06	13.49	8.0E+07
10	15	56000	13	3.5	2.14	5.1E+07	61	8.9E+07	–	–	–	–	1.46	1.37	34	5.4E+07	1.37	0.01	20	2.9E+07	49.74	3.6E+08
15	15	56000	7	3.5	2.78	7.6E+07	68	1.5E+08	–	–	–	–	2.15	2.15	38	8.8E+07	2.13	0.01	21	4.5E+07	119.29	9.3E+08
20	15	56000	3	3.5	2.8	7.1E+07	98	2.8E+08	–	–	–	–	2.89	2.72	37	1.2E+08	2.69	0.01	22	6.4E+07	224.72	1.8E+09
25	15	56000	3	3.5	3.06	7.5E+07	85	3.1E+08	–	–	–	–	3.75	3.23	36	1.4E+08	3.58	0.01	24	8.6E+07	330.67	-2.1E+09

A.7 MiniBooNE Particle Identification

Table A.13: Summary of the results with MiniBooNE Particle Identification ($\times 10^{10}, m = 130064, n = 50$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	8.92236	0.0	0.0	0.0	0.33	1.34	2.51	0.0	267790.23	286945.15	0.02	0.03	0.05	–	–	0.0	0.0	0.0	0.17	0.19	0.21	0.01	267773.14	286912.19	5.17	6.41	7.92	0.0	0.69
3	5.22601	0.0	4.34	21.68	0.47	1.75	2.96	0.0	261272.35	489908.45	0.02	0.07	0.14	–	–	0.0	4.34	21.68	0.26	0.34	0.4	0.02	195956.95	489911.65	7.91	9.27	10.83	21.68	0.88
5	1.82252	0.0	0.0	0.01	0.68	1.89	3.06	0.02	281033.14	1404879.56	0.03	0.3	0.5	–	–	0.01	0.02	0.02	0.51	0.65	0.8	116.92	842989.59	1404928.44	14.33	15.76	18.33	0.0	8.12
10	0.9092	0.01	1.34	8.19	1.92	2.77	3.05	0.16	187742.7	2816058.65	0.04	1.33	1.95	–	–	0.02	0.36	1.92	1.1	1.74	2.53	0.9	563248.17	2816227.27	26.5	29.83	33.63	1.63	44.6
15	0.63506	0.1	2.96	4.94	3.0	3.11	3.82	2.41	4.32	5.15	2.19	2.77	4.0	–	–	0.07	2.07	4.77	1.68	2.29	2.99	3.71	268787.59	4031720.36	39.29	44.77	53.42	0.0	89.06
20	0.50863	0.04	2.01	8.45	3.03	3.32	4.31	7.38	8.31	8.76	2.26	3.52	4.49	–	–	0.03	3.1	9.6	2.51	3.41	4.93	8.92	335603.13	5033890.22	52.67	57.0	61.62	1.17	153.2
25	0.44425	-0.39	2.1	9.45	3.07	4.33	5.92	9.05	9.98	10.42	4.28	5.92	8.46	–	–	-0.43	1.91	9.93	3.31	4.09	5.48	5.8	384238.18	5763383.46	68.45	71.97	78.54	-0.0	218.08
Mean:			**1.82**			**2.65**			**142551.57**			**1.99**		–	–		**1.68**			**1.81**			**408370.96**			**33.57**		**3.5**	**73.52**

Table A.14: Clustering details with MiniBooNE Particle Identification

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	15	130063	17	3.0	1.34	9.6E+06	2	6.2E+05	–	–	–	–	0.17	0.02	2	1.0E+06	6.41	0.0	3	7.8E+05	0.69	1.7E+06
3	15	130063	20	3.0	1.75	1.8E+07	6	2.4E+06	–	–	–	–	0.25	0.08	7	3.5E+06	9.27	0.0	5	2.1E+06	0.88	3.7E+06
5	15	130063	15	3.0	1.89	2.9E+07	21	1.4E+07	–	–	–	–	0.42	0.22	13	1.0E+07	15.76	0.0	5	3.5E+06	8.12	6.2E+07
10	15	130063	12	3.0	2.77	7.2E+07	56	7.3E+07	–	–	–	–	0.87	0.87	34	4.8E+07	29.82	0.01	17	2.2E+07	44.6	4.4E+08
15	15	130063	6	3.0	3.11	8.9E+07	84	1.6E+08	–	–	–	–	1.26	1.03	30	6.5E+07	44.75	0.02	24	4.6E+07	89.06	8.9E+08
20	15	130063	2	3.0	3.32	9.5E+07	87	2.3E+08	–	–	–	–	1.72	1.69	38	1.1E+08	56.97	0.03	25	6.5E+07	153.2	1.6E+09
25	15	130063	1	3.0	4.33	1.4E+08	120	3.9E+08	–	–	–	–	1.97	2.12	40	1.4E+08	71.93	0.03	25	8.2E+07	218.08	-2.1E+09

A.8 MiniBooNE Particle Identification (normalized)

Table A.15: Summary of the results with MiniBooNE Particle Identification (normalized) ($\times 10^2, m = 130064, n = 50$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	28.01938*	0.0	0.02	0.05	0.05	0.6	0.99	0.0	377.13	690.53	0.03	0.1	0.25	–	–	0.0	103.29	690.54	0.18	0.21	0.34	0.25	652.79	693.78	0.52	0.72	1.01	0.0	1.02
3	19.85673*	0.01	2.13	7.07	0.15	0.56	1.0	0.0	246.06	975.91	0.07	0.11	0.18	–	–	0.0	2.8	6.99	0.32	0.38	0.43	0.47	539.64	989.4	0.85	1.04	1.26	6.99	3.1
5	12.10267*	0.04	0.66	3.94	0.25	0.66	1.01	-0.0	0.78	3.85	0.1	0.2	0.35	–	–	-0.0	1.54	3.85	0.59	0.72	1.07	0.84	168.38	1651.07	1.27	1.57	2.16	8.09	10.6
10	8.57382*	0.16	0.72	3.5	0.17	0.67	1.04	0.0	1.37	3.15	0.35	0.8	1.68	–	–	0.01	0.98	3.31	1.21	1.53	2.28	3.14	8.15	13.27	2.31	2.93	3.51	5.25	25.06
15	7.24131*	0.4	1.06	2.19	0.15	0.5	1.0	0.01	1.06	4.26	0.83	1.58	2.19	–	–	0.02	0.71	2.12	1.99	2.55	3.64	4.31	6.79	13.11	3.95	4.43	5.49	0.68	46.13
20	6.30493*	0.35	1.49	4.05	0.37	0.75	1.14	0.04	1.36	2.52	0.94	1.84	2.98	–	–	0.26	1.3	2.45	2.71	3.76	6.09	6.61	9.44	15.68	5.69	6.29	7.71	0.22	67.66
25	5.71335*	0.48	1.18	2.39	0.45	0.82	1.1	0.1	0.68	2.62	1.28	2.48	4.84	–	–	0.02	0.46	1.02	3.61	4.44	6.46	5.89	9.73	12.99	6.64	7.55	8.61	0.47	95.35
Mean:			**1.03**			**0.65**			**89.78**			**1.02**		–	–		**15.87**			**1.94**			**199.27**			**3.51**		**3.1**	**35.56**

Table A.16: Clustering details with MiniBooNE Particle Identification (normalized)

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	20	12000	123	1.0	0.6	5.9E+06	9	2.4E+06	–	–	–	–	0.18	0.03	3	1.3E+06	0.72	0.0	4	1.1E+06	1.02	1.3E+07
3	20	12000	76	1.0	0.56	9.0E+06	9	3.6E+06	–	–	–	–	0.26	0.12	8	4.1E+06	1.04	0.0	6	2.3E+06	3.1	3.4E+07
5	20	12000	62	1.0	0.66	1.4E+07	14	9.4E+06	–	–	–	–	0.46	0.26	16	1.2E+07	1.57	0.0	7	4.8E+06	10.6	1.1E+08
10	20	12000	19	1.0	0.67	2.0E+07	36	4.6E+07	–	–	–	–	0.86	0.66	27	3.9E+07	2.93	0.0	9	1.2E+07	25.06	2.5E+08
15	20	12000	6	1.0	0.5	1.7E+07	46	8.9E+07	–	–	–	–	1.33	1.22	35	7.4E+07	4.43	0.0	11	2.2E+07	46.13	4.7E+08
20	20	12000	5	1.0	0.75	2.6E+07	46	1.2E+08	–	–	–	–	1.75	2.01	45	1.3E+08	6.28	0.01	12	3.2E+07	67.66	7.3E+08
25	20	12000	4	1.0	0.82	3.1E+07	47	1.5E+08	–	–	–	–	2.06	2.39	45	1.6E+08	7.54	0.01	13	4.1E+07	95.35	1.0E+09

A.9 MFCCs for Speech Emotion Recognition

Table A.17: Summary of the results with MFCCs for Speech Emotion Recognition ($\times 10^9, m = 85134, n = 58$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	0.74513*	0.01	0.04	0.07	0.06	0.45	0.94	0.0	0.0	0.01	0.04	0.11	0.17	–	–	0.0	0.0	0.01	0.12	0.19	0.29	0.26	4.78	72.61	1.57	2.17	2.58	0.0	6.85
3	0.50215*	0.02	0.05	0.13	0.04	0.47	1.01	0.0	0.0	0.01	0.06	0.11	0.19	–	–	0.0	0.0	0.01	0.21	0.26	0.32	0.31	6.26	23.11	2.75	3.22	3.75	0.0	8.87
5	0.3456*	0.03	0.06	0.12	0.09	0.51	0.99	0.0	0.01	0.02	0.13	0.24	0.43	–	–	0.0	0.48	9.27	0.38	0.51	0.68	0.55	7.64	26.39	4.69	5.44	6.66	-0.0	19.44
10	0.21763*	0.06	1.73	3.4	0.24	0.71	1.05	0.01	1.96	3.38	0.29	0.47	0.79	–	–	0.01	1.82	4.66	0.95	1.15	1.52	1.63	7.92	19.79	8.8	9.91	11.31	-0.01	40.39
15	0.17608*	0.14	1.23	5.24	0.27	0.63	0.99	1.47	3.39	5.2	0.65	1.01	2.32	–	–	-0.0	0.92	2.15	1.16	1.53	2.1	3.15	8.05	14.48	13.02	15.2	17.99	1.17	69.97
20	0.15383*	0.24	1.24	3.64	0.3	0.74	1.02	0.54	2.55	5.1	0.99	1.78	2.75	–	–	0.02	1.29	2.63	1.64	2.42	3.22	4.72	11.81	20.85	18.61	20.81	23.28	2.55	97.16
25	0.14109*	0.45	1.6	4.71	0.23	0.69	1.01	0.94	4.27	6.57	1.29	2.64	4.86	–	–	0.26	0.97	2.35	2.24	2.89	3.65	6.3	11.61	21.04	21.74	25.21	27.78	0.74	128.27
Mean:		**0.85**			**0.6**			**1.74**			**0.91**			–	–	**0.78**			**1.28**			**8.3**			**11.71**			**0.64**	**52.99**

Table A.18: Clustering details with MFCCs for Speech Emotion Recognition

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	20	12000	53	1.0	0.45	4.1E+06	12	2.1E+06	–	–	–	–	0.1	0.09	10	2.0E+06	2.17	0.0	5	9.0E+05	6.85	2.4E+07
3	20	12000	52	1.0	0.47	6.5E+06	13	3.3E+06	–	–	–	–	0.15	0.11	11	3.4E+06	3.21	0.0	12	3.0E+06	8.87	4.9E+07
5	20	12000	34	1.0	0.51	9.4E+06	22	9.4E+06	–	–	–	–	0.28	0.23	18	8.6E+06	5.43	0.0	12	5.3E+06	19.44	1.5E+08
10	20	12000	22	1.0	0.71	1.8E+07	27	2.3E+07	–	–	–	–	0.66	0.49	24	2.3E+07	9.9	0.01	14	1.2E+07	40.39	3.7E+08
15	20	12000	10	1.0	0.63	1.7E+07	39	5.0E+07	–	–	–	–	0.77	0.77	29	4.0E+07	15.19	0.01	16	2.0E+07	69.97	7.0E+08
20	20	12000	8	1.0	0.74	2.1E+07	55	9.4E+07	–	–	–	–	1.28	1.13	33	6.1E+07	20.8	0.02	18	3.1E+07	97.16	1.0E+09
25	20	12000	5	1.0	0.69	2.1E+07	64	1.4E+08	–	–	–	–	1.55	1.33	33	7.7E+07	25.18	0.03	21	4.6E+07	128.27	1.4E+09

A.10 ISOLET

Table A.19: Summary of the results with ISOLET ($\times 10^5, m = 7797, n = 617$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	7.2194	0.02	0.03	0.05	0.29	2.88	5.65	0.0	0.0	0.01	0.02	0.03	0.06	0.0	17.89	0.0	0.0	0.0	0.17	0.2	0.22	1.92	4.61	25.2	0.96	1.29	1.73	-0.0	7.51
3	6.78782	0.03	0.23	0.62	0.5	3.39	5.88	0.0	0.55	1.85	0.06	0.09	0.18	0.56	17.91	0.0	0.59	2.75	0.28	0.33	0.45	2.87	4.57	6.67	1.31	1.76	1.99	0.0	16.01
5	6.13651	0.05	0.71	1.76	1.5	3.68	5.89	0.0	1.08	2.88	0.06	0.2	0.48	0.01	17.94	0.0	0.48	1.86	0.47	0.61	0.78	4.36	6.12	8.6	2.2	2.56	3.03	0.0	36.11
10	5.28577	0.13	0.89	2.16	1.2	3.48	5.91	0.01	1.33	3.41	0.23	0.4	0.62	0.75	17.95	0.01	0.65	2.17	0.96	1.08	1.28	4.37	7.42	11.97	4.81	5.13	5.68	2.84	68.27
15	4.87391	0.22	1.28	2.59	1.16	3.82	5.94	0.03	1.86	3.94	0.32	0.51	0.92	-0.08	18.01	0.19	1.11	1.85	1.24	1.56	1.91	4.74	7.32	11.88	6.51	7.35	8.12	0.54	110.34
20	4.60857	0.42	1.07	2.82	1.11	3.57	6.11	0.1	1.7	4.5	0.39	0.8	1.31	-0.15	18.08	0.03	1.42	3.06	1.75	2.07	2.54	5.77	7.86	9.22	9.1	9.53	10.12	0.03	155.11
25	4.44323	0.11	0.93	2.51	1.39	3.85	6.01	0.75	1.63	2.74	0.57	0.84	1.41	-0.38	18.09	-0.05	0.83	2.35	2.08	2.47	2.81	6.15	7.77	12.35	11.0	11.69	12.11	0.3	209.38
Mean:		**0.73**			**3.52**			**1.16**			**0.41**			**0.1**	**17.98**	**0.73**			**1.19**			**6.52**			**5.62**			**0.53**	**86.1**

Table A.20: Clustering details with ISOLET

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	15	4000	166	6.0	2.88	4.0E+06	7	1.0E+05	17.87	0.02	4	3.0E+07	0.15	0.05	7	1.4E+05	1.29	0.0	4	6.8E+04	7.51	7.6E+06
3	15	4000	146	6.0	3.39	5.5E+06	11	2.6E+05	17.87	0.04	4	3.0E+07	0.24	0.09	11	3.1E+05	1.75	0.0	6	1.3E+05	16.01	1.6E+07
5	15	4000	108	6.0	3.68	7.4E+06	17	6.7E+05	17.87	0.07	6	3.1E+07	0.4	0.21	18	7.8E+05	2.56	0.01	7	2.9E+05	36.11	3.7E+07
10	15	4000	54	6.0	3.48	9.5E+06	20	1.6E+06	17.87	0.08	4	3.1E+07	0.71	0.37	17	1.5E+06	5.12	0.01	10	7.6E+05	68.27	7.3E+07
15	15	4000	40	6.0	3.82	1.2E+07	18	2.1E+06	17.87	0.14	5	3.1E+07	1.06	0.5	16	2.2E+06	7.32	0.04	12	1.5E+06	110.34	1.2E+08
20	15	4000	23	6.0	3.57	1.1E+07	22	3.5E+06	17.87	0.21	6	3.1E+07	1.42	0.66	16	3.0E+06	9.48	0.05	12	1.9E+06	155.11	1.7E+08
25	15	4000	20	6.0	3.85	1.2E+07	19	3.7E+06	17.87	0.22	5	3.1E+07	1.63	0.84	17	4.0E+06	11.63	0.06	13	2.5E+06	209.38	2.3E+08

A.11 Sensorless Drive Diagnosis

Table A.21: Summary of the results with Sensorless Drive Diagnosis ($\times 10^7, m = 58509, n = 48$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	3.88116	-0.0	10.02	100.19	0.09	0.57	1.01	100.2	100.22	100.23	0.03	0.05	0.08	-0.0	295.71	-0.0	22.55	100.23	0.06	0.06	0.1	100.26	100.76	103.61	0.68	0.96	1.28	-0.0	0.36
3	2.91313	-0.0	6.4	16.97	0.13	0.56	1.02	155.22	156.41	157.46	0.03	0.09	0.14	-0.0	295.74	-0.0	5.88	27.03	0.09	0.11	0.19	155.46	156.98	159.97	1.05	1.36	1.69	-0.0	0.48
5	1.93651	0.0	4.44	37.85	0.4	0.79	1.02	37.87	55.32	277.27	0.11	0.23	0.37	0.02	295.86	0.01	2.18	6.21	0.16	0.27	0.56	38.25	181.23	268.9	1.53	2.09	2.47	5.32	2.04
10	0.98472	-2.42	4.81	23.18	0.5	0.88	1.03	127.26	128.24	129.59	0.21	0.47	0.76	-2.21	295.8	-2.4	5.88	23.18	0.38	0.53	0.76	128.89	131.23	134.79	3.52	3.91	4.58	1.89	7.56
15	0.62816	0.01	1.43	23.44	0.98	1.07	1.28	235.09	236.06	238.04	0.31	0.46	0.78	0.32	295.99	0.02	1.5	30.73	0.63	0.91	1.25	203.4	238.25	243.57	5.09	6.15	8.27	-0.0	28.88
20	0.49884	-0.6	1.95	5.81	1.0	1.22	1.59	260.02	307.48	311.87	0.5	0.69	0.89	-0.58	296.08	-0.58	1.55	7.13	0.92	1.13	1.56	120.98	303.5	316.74	7.15	8.1	9.64	1.01	42.89
25	0.42225	0.45	3.03	9.28	1.14	1.52	2.06	315.13	345.7	376.66	0.64	1.12	1.68	1.28	295.91	0.89	2.68	9.47	1.01	1.4	1.79	319.39	373.35	383.08	8.63	9.76	12.1	0.02	64.52
Mean:			**4.58**			**0.94**			**189.92**			**0.45**		**-0.17**	**295.87**		**6.03**			**0.63**			**212.18**			**4.62**		**1.18**	**20.96**

Table A.22: Clustering details with Sensorless Drive Diagnosis

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	40	58508	24	1.0	0.57	6.0E+06	10	1.1E+06	295.7	0.01	2	1.7E+09	0.05	0.01	3	6.2E+05	0.96	0.0	7	7.8E+05	0.36	2.2E+06
3	40	58508	20	1.0	0.56	7.9E+06	15	2.7E+06	295.73	0.01	2	1.7E+09	0.09	0.03	5	1.3E+06	1.36	0.0	9	1.6E+06	0.48	2.9E+06
5	40	58508	19	1.0	0.79	1.6E+07	32	9.3E+06	295.76	0.1	13	1.7E+09	0.16	0.12	16	5.3E+06	2.08	0.0	16	4.6E+06	2.04	1.7E+07
10	40	58508	9	1.0	0.88	2.4E+07	43	2.5E+07	295.7	0.1	9	1.7E+09	0.3	0.24	19	1.3E+07	3.9	0.01	23	1.4E+07	7.56	6.9E+07
15	40	58508	3	1.0	1.07	3.1E+07	31	2.7E+07	295.7	0.29	15	1.7E+09	0.42	0.48	29	2.8E+07	6.14	0.01	22	1.9E+07	28.88	3.1E+08
20	40	58508	1	1.0	1.22	3.6E+07	35	4.1E+07	295.7	0.38	20	1.7E+09	0.6	0.54	26	3.4E+07	8.09	0.01	23	2.7E+07	42.89	4.9E+08
25	40	58508	1	1.0	1.52	4.6E+07	51	7.4E+07	295.69	0.22	10	1.7E+09	0.74	0.66	26	4.3E+07	9.74	0.02	22	3.2E+07	64.52	7.2E+08

A.12 Sensorless Drive Diagnosis (normalized)

Table A.23: Summary of the results with Sensorless Drive Diagnosis (normalized) ($\times 10^3, m = 58509, n = 48$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	15.64798*	0.02	0.09	0.25	0.02	0.16	0.29	0.0	0.0	0.01	0.02	0.03	0.05	0.0	213.66	0.0	1.89	75.65	0.09	0.11	0.25	0.35	19.26	107.3	0.27	0.36	0.46	106.83	0.67
3	12.19375*	0.05	1.71	3.7	0.05	0.2	0.3	0.0	1.85	9.37	0.03	0.06	0.19	0.0	213.67	0.0	3.04	10.33	0.11	0.15	0.28	0.86	10.69	30.59	0.4	0.53	0.72	28.29	2.95
5	7.85054*	0.09	0.64	10.92	0.02	0.14	0.3	0.0	1.2	9.17	0.04	0.08	0.31	0.19	213.69	0.0	1.87	10.95	0.17	0.22	0.31	0.52	20.93	73.71	0.57	0.81	1.06	36.22	5.02
10	4.71275*	0.25	4.02	11.27	0.03	0.2	0.3	0.0	6.48	31.36	0.08	0.21	0.51	0.32	213.69	0.32	4.78	9.64	0.33	0.46	0.66	6.88	33.32	54.58	1.04	1.45	2.23	14.07	13.66
15	3.62541*	0.73	4.52	10.62	0.05	0.2	0.31	2.92	8.78	18.21	0.15	0.26	0.45	0.35	213.71	0.61	5.07	12.74	0.52	0.67	0.84	19.52	39.5	82.68	1.74	2.26	3.14	12.75	20.58
20	2.971*	0.9	4.84	10.5	0.06	0.22	0.3	2.74	13.97	26.53	0.21	0.39	0.76	-0.02	213.73	1.09	6.13	13.02	0.72	0.98	1.35	40.37	84.75	135.27	2.21	2.91	3.46	7.49	28.98
25	2.60929*	0.82	5.14	10.56	0.05	0.2	0.33	4.25	14.2	26.47	0.22	0.46	0.77	1.46	213.76	0.69	4.76	16.82	0.91	1.16	1.59	50.89	84.72	149.74	2.58	3.37	4.05	12.17	36.47
Mean:			**2.99**			**0.19**			**6.64**			**0.21**		**0.33**	**213.7**		**3.93**			**0.54**			**41.88**			**1.67**		**31.12**	**15.48**

Table A.24: Clustering details with Sensorless Drive Diagnosis (normalized)

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	40	3500	55	0.3	0.16	1.2E+06	6	7.5E+05	213.64	0.02	4	1.7E+09	0.08	0.04	6	9.6E+05	0.36	0.0	5	5.3E+05	0.67	8.4E+06
3	40	3500	76	0.3	0.2	2.8E+06	12	2.1E+06	213.65	0.02	5	1.7E+09	0.09	0.06	11	2.4E+06	0.53	0.0	6	1.1E+06	2.95	4.2E+07
5	40	3500	43	0.3	0.14	2.8E+06	13	3.8E+06	213.64	0.06	8	1.7E+09	0.15	0.07	10	3.6E+06	0.81	0.0	9	2.7E+06	5.02	7.1E+07
10	40	3500	27	0.3	0.2	4.9E+06	20	1.2E+07	213.63	0.06	5	1.7E+09	0.28	0.18	16	1.1E+07	1.45	0.0	12	6.9E+06	13.66	2.0E+08
15	40	3500	17	0.3	0.2	5.9E+06	18	1.6E+07	213.64	0.07	6	1.7E+09	0.43	0.24	15	1.5E+07	2.25	0.01	12	1.1E+07	20.58	3.0E+08
20	40	3500	14	0.3	0.22	7.0E+06	20	2.3E+07	213.64	0.09	5	1.7E+09	0.59	0.38	18	2.4E+07	2.9	0.01	17	2.0E+07	28.98	4.2E+08
25	40	3500	8	0.3	0.2	6.5E+06	20	3.0E+07	213.63	0.12	6	1.7E+09	0.71	0.45	17	2.9E+07	3.36	0.01	19	2.7E+07	36.47	5.1E+08

A.13 Online News Popularity

Table A.25: Summary of the results with Online News Popularity ($\times 10^{14}, m = 39644, n = 58$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	9.53913	0.0	0.01	0.04	0.05	0.4	0.7	-0.0	-0.0	-0.0	0.01	0.03	0.06	-0.0	101.96	-0.0	17.5	174.95	0.05	0.07	0.13	0.02	47.74	192.35	2.47	2.99	3.37	-0.0	2.69
3	5.91077	0.01	4.85	24.1	0.09	0.43	0.66	0.02	8.54	61.15	0.01	0.04	0.08	0.02	101.99	0.0	7.84	49.62	0.08	0.12	0.17	0.04	94.52	371.8	3.44	4.18	4.68	0.0	38.51
5	3.09885	0.04	3.73	31.06	0.08	0.47	0.7	0.0	10.47	33.54	0.03	0.08	0.15	0.0	102.01	0.0	6.98	30.96	0.15	0.2	0.28	114.51	197.04	207.86	6.05	6.91	8.09	0.0	55.66
10	1.17247	0.29	3.97	17.66	0.13	0.39	0.71	0.03	17.63	49.89	0.09	0.31	0.7	0.02	102.02	0.0	3.11	14.72	0.33	0.49	0.85	402.53	642.35	711.96	12.41	14.04	16.13	2.57	79.54
15	0.77637	0.81	5.12	16.32	0.16	0.49	0.89	9.76	16.71	22.47	0.22	0.54	0.85	0.01	102.14	0.01	4.45	15.59	0.5	0.67	1.02	452.41	863.87	1125.94	17.85	20.03	23.22	14.77	98.24
20	0.59809	2.68	5.96	10.62	0.13	0.44	0.74	1.38	28.31	38.82	0.3	0.74	1.16	0.26	102.21	0.94	3.99	11.15	0.72	0.92	1.37	302.79	1034.27	1487.14	24.5	26.6	29.74	7.43	121.45
25	0.49616	3.07	6.24	10.59	0.16	0.46	0.75	14.05	36.25	59.03	0.55	1.01	1.63	0.39	102.26	1.49	5.05	10.26	0.85	1.11	1.36	625.99	1367.73	1673.04	29.16	32.02	37.71	7.03	146.78
Mean:		**4.27**			**0.44**			**16.84**			**0.39**			**0.1**	**102.08**	**6.99**			**0.51**			**606.79**			**15.25**			**4.54**	**77.55**

Table A.26: Clustering details with Online News Popularity

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	20	10000	87	0.7	0.4	3.6E+06	7	5.7E+05	101.96	0.01	2	7.9E+08	0.05	0.02	4	4.6E+05	2.99	0.0	4	3.2E+05	2.69	3.2E+07
3	20	10000	53	0.7	0.43	5.9E+06	9	1.1E+06	101.96	0.03	8	7.9E+08	0.08	0.04	7	1.1E+06	4.18	0.0	4	5.0E+05	38.51	3.6E+08
5	20	10000	48	0.7	0.47	8.9E+06	15	3.0E+06	101.95	0.06	10	7.9E+08	0.13	0.08	12	2.9E+06	6.9	0.0	5	1.0E+06	55.66	5.4E+08
10	20	10000	16	0.7	0.39	9.0E+06	32	1.3E+07	101.96	0.06	6	7.9E+08	0.25	0.24	21	9.6E+06	14.04	0.0	6	2.6E+06	79.54	7.5E+08
15	20	10000	9	0.7	0.49	1.4E+07	41	2.4E+07	101.95	0.19	12	7.9E+08	0.37	0.29	20	1.3E+07	20.02	0.01	8	4.7E+06	98.24	9.3E+08
20	20	10000	6	0.7	0.44	1.2E+07	45	3.6E+07	101.97	0.24	16	8.0E+08	0.49	0.43	24	2.1E+07	26.59	0.01	8	6.8E+06	121.45	1.2E+09
25	20	10000	4	0.7	0.46	1.3E+07	51	5.0E+07	101.96	0.29	14	8.0E+08	0.6	0.5	23	2.6E+07	32.01	0.01	9	8.7E+06	146.78	1.4E+09

A.14 Gas Sensor Array Drift

Table A.27: Summary of the results with Gas Sensor Array Drift ($\times 10^{13}, m = 13910, n = 128$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	7.91186	0.07	0.17	0.46	0.26	4.61	7.89	-0.0	0.0	0.0	0.01	0.03	0.03	-0.0	15.94	-0.0	0.0	0.0	0.07	0.08	0.13	0.03	0.17	0.5	1.8	2.22	2.59	-0.0	42.33
3	5.02412	0.07	0.2	0.45	0.21	3.93	7.43	0.0	0.0	0.01	0.02	0.05	0.07	0.0	15.95	-0.0	1.1	32.96	0.1	0.13	0.22	0.02	0.5	5.16	2.71	3.28	3.93	-0.0	48.34
5	3.22394	0.03	4.59	8.4	0.96	3.73	7.58	6.83	8.05	8.13	0.04	0.08	0.11	0.1	16.01	0.01	3.66	8.13	0.14	0.2	0.31	0.15	8.56	38.8	4.44	5.28	5.96	-0.0	51.03
10	1.65524	-0.11	3.91	22.65	0.33	4.24	7.74	-0.0	40.7	59.32	0.11	0.26	0.46	-0.16	16.0	-0.16	4.28	21.91	0.29	0.34	0.5	5.3	22.75	60.64	8.93	10.26	11.69	-0.0	69.71
15	1.13801	-0.74	5.38	13.99	0.52	3.79	7.8	18.54	52.85	101.14	0.21	0.5	0.93	-0.2	16.01	-0.83	4.42	9.59	0.48	0.6	0.76	7.69	35.79	104.04	13.67	15.11	16.81	0.35	82.86
20	0.87916	0.12	4.0	9.99	1.1	4.41	8.09	17.58	39.66	54.86	0.22	0.69	1.43	1.55	16.03	0.58	3.86	7.5	0.56	0.78	1.35	10.0	29.66	58.29	17.51	19.96	22.55	2.69	95.35
25	0.72274	0.42	4.32	12.46	0.59	4.52	7.95	21.72	50.44	80.74	0.34	1.06	1.98	4.13	16.36	0.64	4.69	8.58	0.77	0.96	1.27	8.78	17.7	40.43	22.67	24.49	28.15	3.01	109.72
Mean:		**3.23**			**4.17**			**27.39**			**0.38**			**0.78**	**16.04**	**3.15**			**0.44**			**16.45**			**11.51**			**0.86**	**71.33**

Table A.28: Clustering details with Gas Sensor Array Drift

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	30	9000	471	8.0	4.61	3.7E+07	12	3.3E+05	15.93	0.01	6	9.7E+07	0.06	0.03	9	3.1E+05	2.22	0.0	6	1.8E+05	42.33	1.7E+08
3	30	9000	388	8.0	3.93	3.7E+07	16	6.8E+05	15.93	0.01	6	9.7E+07	0.09	0.04	10	5.0E+05	3.27	0.0	15	6.1E+05	48.34	2.0E+08
5	30	9000	238	8.0	3.73	4.5E+07	20	1.4E+06	15.94	0.07	19	9.8E+07	0.12	0.08	14	1.2E+06	5.28	0.01	14	9.6E+05	51.03	2.2E+08
10	30	9000	114	8.0	4.24	4.9E+07	27	3.8E+06	15.93	0.07	7	9.8E+07	0.24	0.1	13	2.2E+06	10.24	0.01	17	2.4E+06	69.71	3.0E+08
15	30	9000	74	8.0	3.79	5.1E+07	41	8.6E+06	15.93	0.08	7	9.8E+07	0.38	0.22	19	4.5E+06	15.07	0.04	23	4.9E+06	82.86	3.8E+08
20	30	9000	67	8.0	4.41	6.6E+07	45	1.3E+07	15.93	0.1	7	9.9E+07	0.44	0.34	22	6.9E+06	19.92	0.04	23	6.5E+06	95.35	4.4E+08
25	30	9000	52	8.0	4.52	7.3E+07	60	2.1E+07	15.93	0.43	24	1.1E+08	0.58	0.38	21	8.3E+06	24.43	0.06	25	8.7E+06	109.72	5.0E+08

A.15 3D Road Network

Table A.29: Summary of the results with 3D Road Network ($\times 10^6, m = 434874, n = 3$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	49.13298	0.0	0.01	0.03	0.05	0.28	0.5	0.0	0.0	0.01	0.06	0.21	0.27	–	–	0.0	0.0	0.01	0.07	0.21	0.29	0.0	0.21	0.85	0.43	0.59	0.94	0.0	13.97
3	22.77818	0.0	0.01	0.09	0.08	0.29	0.52	0.0	0.01	0.02	0.11	0.22	0.3	–	–	0.0	0.01	0.02	0.14	0.24	0.33	0.03	3.28	115.72	0.55	0.84	1.18	0.0	15.12
5	8.82574	0.0	0.03	0.12	0.1	0.3	0.5	0.02	0.02	0.03	0.18	0.3	0.41	–	–	0.01	0.02	0.03	0.2	0.31	0.44	0.05	4.76	55.28	0.87	1.24	1.71	0.0	19.14
10	2.56661	0.01	0.22	1.01	0.09	0.38	0.62	0.08	0.34	0.37	0.3	0.94	1.31	–	–	0.03	0.33	0.47	0.34	0.63	1.08	0.74	12.53	63.59	1.4	2.06	3.03	0.0	30.9
15	1.27069	0.05	0.48	1.51	0.17	0.44	0.9	0.05	0.4	0.77	0.69	2.3	2.91	–	–	0.05	0.57	1.21	0.63	0.97	2.05	2.37	21.71	52.12	2.31	3.01	4.26	0.0	48.65
20	0.80865	0.06	1.13	3.18	0.19	0.47	0.83	2.56	5.65	8.94	1.67	2.86	3.39	–	–	0.07	1.45	7.18	0.75	1.18	2.1	5.64	31.63	57.88	3.04	4.4	6.11	-0.0	69.55
25	0.59259	0.33	1.72	4.89	0.3	0.6	1.31	2.35	9.9	19.91	2.3	3.83	4.15	–	–	0.15	1.48	3.85	0.91	1.56	3.06	20.27	48.42	78.7	3.94	4.95	7.16	-0.0	102.5
Mean:			**0.51**			**0.39**			**2.33**			**1.52**		–	–		**0.55**			**0.73**			**17.51**			**2.44**		**-0.0**	**42.83**

Table A.30: Clustering details with 3D Road Network

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	40	100000	17	0.5	0.28	9.2E+06	11	9.7E+06	–	–	–	–	0.04	0.17	9	9.8E+06	0.59	0.0	7	6.3E+06	13.97	2.3E+08
3	40	100000	15	0.5	0.29	1.5E+07	15	2.0E+07	–	–	–	–	0.06	0.18	13	2.0E+07	0.84	0.0	10	1.4E+07	15.12	3.7E+08
5	40	100000	12	0.5	0.3	3.0E+07	34	7.4E+07	–	–	–	–	0.12	0.19	21	5.0E+07	1.24	0.0	16	3.5E+07	19.14	6.3E+08
10	40	100000	6	0.5	0.38	8.1E+07	116	5.0E+08	–	–	–	–	0.23	0.4	43	2.0E+08	2.06	0.0	28	1.2E+08	30.9	1.4E+09
15	40	100000	3	0.5	0.44	1.2E+08	243	1.6E+09	–	–	–	–	0.37	0.6	57	3.9E+08	3.01	0.01	40	2.6E+08	48.65	-2.1E+09
20	40	100000	2	0.5	0.47	1.4E+08	261	2.3E+09	–	–	–	–	0.49	0.69	56	5.1E+08	4.4	0.01	42	3.7E+08	69.55	-2.1E+09
25	40	100000	2	0.5	0.6	2.1E+08	297	3.2E+09	–	–	–	–	0.61	0.95	72	8.1E+08	4.94	0.01	57	6.2E+08	102.5	-2.1E+09

A.16 Skin Segmentation

Table A.31: Summary of the results with Skin Segmentation ($\times 10^9, m = 245057, n = 3$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	1.32236	0.01	0.04	0.09	0.01	0.11	0.2	-0.0	0.0	0.0	0.04	0.08	0.11	–	–	-0.0	0.0	0.0	0.04	0.08	0.15	0.01	0.17	0.79	0.35	0.48	0.7	-0.0	0.93
3	0.89362	0.01	0.07	0.2	0.03	0.1	0.2	-0.0	0.01	0.03	0.03	0.12	0.35	–	–	0.0	0.01	0.02	0.05	0.15	0.3	0.06	11.64	48.07	0.47	0.65	0.81	-0.0	1.72
5	0.50203	0.03	2.03	9.43	0.01	0.12	0.2	0.0	2.98	13.81	0.02	0.06	0.12	–	–	0.0	1.77	13.81	0.07	0.1	0.2	0.06	15.67	97.42	0.66	0.99	1.3	0.0	3.47
10	0.25121	0.12	7.95	21.54	0.03	0.11	0.19	0.02	10.69	38.57	0.04	0.08	0.16	–	–	0.0	5.29	17.35	0.13	0.17	0.28	0.2	34.65	59.38	1.42	1.79	2.51	13.37	7.97
15	0.16964	-1.24	4.29	10.22	0.02	0.12	0.19	2.32	15.78	44.48	0.06	0.13	0.22	–	–	0.05	6.25	17.09	0.22	0.3	0.42	15.83	41.57	88.83	1.87	2.43	3.21	3.84	15.1
20	0.12615	-0.1	4.97	15.05	0.03	0.11	0.17	5.79	21.96	54.92	0.08	0.18	0.31	–	–	0.78	6.88	12.65	0.25	0.37	0.52	28.25	63.6	141.05	2.45	3.31	4.12	4.51	22.33
25	0.10228	2.0	6.11	14.66	0.03	0.11	0.2	8.46	19.98	36.39	0.11	0.24	0.51	–	–	2.6	6.43	12.01	0.3	0.43	0.56	33.63	70.67	113.93	3.07	4.3	5.12	5.78	30.74
Mean:			**3.64**			**0.11**			**10.2**			**0.13**		–	–		**3.81**			**0.23**			**34.0**			**1.99**		**3.93**	**11.75**

Table A.32: Clustering details with Skin Segmentation

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	30	8000	61	0.2	0.11	3.1E+06	8	3.8E+06	–	–	–	–	0.02	0.06	7	4.2E+06	0.47	0.0	5	2.3E+06	0.93	1.2E+08
3	30	8000	53	0.2	0.1	5.1E+06	16	1.2E+07	–	–	–	–	0.03	0.11	14	1.2E+07	0.65	0.0	9	6.6E+06	1.72	2.2E+08
5	30	8000	75	0.2	0.12	1.1E+07	15	1.8E+07	–	–	–	–	0.05	0.06	12	1.8E+07	0.99	0.0	12	1.5E+07	3.47	3.7E+08
10	30	8000	55	0.2	0.11	2.3E+07	19	4.7E+07	–	–	–	–	0.1	0.07	15	4.3E+07	1.79	0.0	14	3.4E+07	7.97	6.6E+08
15	30	8000	38	0.2	0.12	3.4E+07	28	1.0E+08	–	–	–	–	0.17	0.13	17	7.2E+07	2.43	0.0	17	6.4E+07	15.1	1.1E+09
20	30	8000	30	0.2	0.11	3.9E+07	30	1.5E+08	–	–	–	–	0.22	0.14	18	1.0E+08	3.31	0.0	19	9.2E+07	22.33	1.6E+09
25	30	8000	24	0.2	0.11	4.3E+07	36	2.2E+08	–	–	–	–	0.26	0.17	16	1.2E+08	4.29	0.0	24	1.4E+08	30.74	2.1E+09

A.17 KEGG Metabolic Relation Network (Directed)

Table A.33: Summary of the results with KEGG Metabolic Relation Network (Directed) ($\times 10^8, m = 53413, n = 20$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	11.3853	-0.0	3.82	18.87	0.04	0.53	0.9	-0.0	17.92	18.86	0.01	0.04	0.05	-0.0	167.73	-0.0	9.45	18.96	0.02	0.03	0.05	18.86	19.01	19.4	0.4	0.58	0.74	-0.0	0.18
3	4.9006	0.0	0.08	0.56	0.1	0.44	0.98	124.82	124.82	124.83	0.05	0.07	0.09	0.0	167.77	0.0	0.0	0.0	0.04	0.06	0.08	124.82	128.04	176.22	0.64	0.79	1.09	-0.0	0.49
5	1.88367	0.0	3.05	30.39	0.09	0.62	0.97	0.07	2.31	44.91	0.13	0.15	0.17	0.07	167.77	0.0	0.01	0.07	0.08	0.14	0.19	0.05	29.08	434.43	0.94	1.21	1.45	0.0	1.85
10	0.60513	0.0	5.34	39.04	0.3	0.75	1.19	36.82	43.32	91.1	0.34	0.44	0.56	0.01	167.76	0.01	5.5	35.06	0.12	0.18	0.33	38.55	43.13	81.45	1.99	2.23	2.47	4.96	5.88
15	0.35393	-0.86	6.89	18.54	0.43	0.71	0.98	97.62	98.91	109.04	0.91	1.08	1.2	0.75	167.85	-0.54	4.0	12.79	0.22	0.3	0.51	118.63	121.46	125.14	2.91	3.2	5.08	0.71	13.24
20	0.25027	-0.34	4.65	25.28	0.42	0.72	1.03	164.2	170.29	173.33	1.02	1.37	1.61	0.55	167.87	-0.15	3.09	7.29	0.33	0.5	0.76	185.83	201.84	211.57	3.95	4.61	5.16	3.32	18.05
25	0.19289	1.78	3.75	8.26	0.48	0.87	1.12	235.59	247.13	251.35	1.13	1.43	2.12	1.35	167.88	0.44	4.31	9.32	0.34	0.51	0.78	261.84	275.72	299.44	4.56	5.18	5.99	1.64	25.33
Mean:		**3.94**			**0.66**			**100.67**			**0.66**			**0.39**	**167.8**	**3.77**			**0.25**			**116.9**			**2.54**			**1.52**	**9.29**

Table A.34: Clustering details with KEGG Metabolic Relation Network (Directed)

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	20	53350	52	1.0	0.53	1.1E+07	12	1.3E+06	167.73	0.0	2	1.4E+09	0.02	0.01	5	7.2E+05	0.58	0.0	9	9.5E+05	0.18	8.9E+05
3	20	53350	35	1.0	0.44	1.3E+07	21	3.4E+06	167.73	0.04	11	1.4E+09	0.03	0.03	10	2.0E+06	0.78	0.0	16	2.6E+06	0.49	6.2E+06
5	20	53350	41	1.0	0.62	2.8E+07	39	1.0E+07	167.73	0.05	15	1.4E+09	0.05	0.09	21	6.3E+06	1.2	0.01	33	8.8E+06	1.85	3.0E+07
10	20	53350	28	1.0	0.75	4.7E+07	87	4.6E+07	167.73	0.03	8	1.4E+09	0.08	0.1	23	1.4E+07	2.22	0.01	71	3.8E+07	5.88	1.0E+08
15	20	53350	18	1.0	0.71	4.8E+07	150	1.2E+08	167.73	0.12	15	1.4E+09	0.16	0.14	18	1.7E+07	3.18	0.02	72	5.7E+07	13.24	2.5E+08
20	20	53350	11	1.0	0.72	5.5E+07	158	1.7E+08	167.72	0.14	15	1.4E+09	0.19	0.31	32	3.7E+07	4.58	0.03	84	9.0E+07	18.05	3.5E+08
25	20	53350	12	1.0	0.87	6.8E+07	143	1.9E+08	167.72	0.15	14	1.4E+09	0.22	0.28	25	3.7E+07	5.14	0.04	92	1.2E+08	25.33	5.1E+08

A.18 Shuttle Control

Table A.35: Summary of the results with Shuttle Control ($\times 10^8, m = 58000, n = 9$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	21.34329	0.0	5.76	51.12	0.06	0.45	0.94	51.06	51.1	51.14	0.01	0.02	0.03	0.0	264.62	0.0	2.35	5.04	0.01	0.02	0.03	51.08	51.15	51.21	0.22	0.28	0.39	0.0	0.14
3	10.85415	0.0	6.75	91.71	0.27	0.61	0.98	90.76	144.26	195.93	0.01	0.03	0.07	0.0	264.61	0.0	11.55	91.64	0.02	0.02	0.03	90.78	163.4	196.34	0.29	0.4	0.54	0.0	0.24
4	8.8691	0.0	2.88	15.19	0.15	0.52	0.95	15.2	124.88	261.12	0.01	0.03	0.06	5.28	264.61	-0.0	3.01	15.21	0.02	0.02	0.03	15.23	138.74	260.7	0.42	0.52	0.61	-0.0	0.39
5	7.24479	0.0	5.52	23.56	0.11	0.56	0.98	38.73	98.19	197.25	0.03	0.05	0.08	1.48	264.62	0.09	10.82	38.8	0.03	0.04	0.08	38.71	132.84	197.64	0.49	0.62	0.72	0.09	0.55
10	2.83216	0.27	7.75	21.06	0.14	0.46	0.96	91.68	157.41	248.3	0.04	0.07	0.12	0.93	264.61	0.35	7.09	24.31	0.06	0.08	0.18	91.02	169.23	247.35	0.89	1.0	1.2	0.55	2.54
15	1.53154	1.33	12.63	40.74	0.18	0.61	1.0	190.13	256.98	327.2	0.05	0.09	0.18	-0.0	264.64	1.31	16.08	40.42	0.11	0.19	0.28	190.04	272.42	402.07	1.29	1.48	1.65	0.02	5.1
20	1.06012	-1.48	11.49	43.38	0.24	0.61	0.96	270.43	319.13	496.82	0.07	0.13	0.17	-3.34	264.65	-3.48	5.22	28.32	0.14	0.21	0.34	284.76	334.93	413.67	1.6	2.12	2.55	0.03	8.22
25	0.77978	2.04	6.81	12.47	0.38	0.66	1.0	369.02	411.67	437.66	0.1	0.16	0.26	-0.09	264.65	3.09	8.04	22.92	0.15	0.22	0.34	365.83	423.2	622.91	1.91	2.41	2.87	-0.0	12.71
Mean:		**7.45**			**0.56**			**195.45**			**0.07**			**0.53**	**264.62**	**8.02**			**0.1**			**210.74**			**1.11**			**0.09**	**3.74**

Table A.36: Clustering details with Shuttle Control

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	15	57950	71	1.0	0.45	1.7E+07	7	8.1E+05	264.61	0.0	2	1.7E+09	0.01	0.0	3	5.3E+05	0.28	0.0	6	6.5E+05	0.14	8.1E+05
3	15	57950	98	1.0	0.61	3.5E+07	11	2.0E+06	264.6	0.01	5	1.7E+09	0.01	0.01	3	8.9E+05	0.4	0.0	8	1.3E+06	0.24	2.4E+06
4	15	57950	82	1.0	0.52	3.9E+07	14	3.2E+06	264.6	0.0	2	1.7E+09	0.02	0.01	3	1.3E+06	0.51	0.0	12	2.7E+06	0.39	3.6E+06
5	15	57950	85	1.0	0.56	5.1E+07	17	4.9E+06	264.61	0.0	2	1.7E+09	0.03	0.01	4	1.9E+06	0.61	0.0	13	3.7E+06	0.55	5.2E+06
10	15	57950	48	1.0	0.46	6.2E+07	20	1.2E+07	264.6	0.01	5	1.7E+09	0.06	0.03	7	5.7E+06	1.0	0.0	16	9.1E+06	2.54	2.9E+07
15	15	57950	38	1.0	0.61	8.6E+07	20	1.8E+07	264.61	0.04	8	1.7E+09	0.12	0.07	10	1.1E+07	1.48	0.0	18	1.5E+07	5.1	7.2E+07
20	15	57950	28	1.0	0.61	9.3E+07	23	2.7E+07	264.62	0.04	6	1.7E+09	0.11	0.1	13	1.9E+07	2.12	0.01	23	2.6E+07	8.22	1.4E+08
25	15	57950	24	1.0	0.66	1.0E+08	24	3.5E+07	264.6	0.05	9	1.7E+09	0.13	0.09	13	2.3E+07	2.41	0.01	23	3.3E+07	12.71	2.4E+08

A.19 Shuttle Control (normalized)

Table A.37: Summary of the results with Shuttle Control (normalized) ($\times 10^1, m = 58000, n = 9$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	104.41601	0.05	0.23	0.72	0.03	0.12	0.2	0.0	13.89	43.49	0.01	0.02	0.04	0.0	264.97	0.0	3.68	14.75	0.01	0.02	0.04	0.1	22.68	46.45	0.03	0.05	0.08	0.0	2.77
3	73.28769	0.08	1.05	2.56	0.01	0.11	0.19	0.0	5.06	28.04	0.02	0.03	0.04	0.02	264.99	0.0	2.08	28.04	0.02	0.03	0.06	0.07	16.91	34.29	0.04	0.06	0.09	2.13	2.98
4	50.076	0.14	8.61	33.21	0.02	0.11	0.19	0.0	1.24	24.83	0.01	0.03	0.06	0.0	264.97	0.0	9.49	31.48	0.03	0.04	0.07	0.14	8.69	33.83	0.06	0.08	0.12	0.01	4.2
5	39.78043	0.15	2.96	12.27	0.01	0.11	0.19	0.0	3.9	11.39	0.02	0.03	0.06	0.0	264.98	0.0	2.36	11.39	0.03	0.05	0.11	1.36	11.21	36.76	0.06	0.09	0.13	0.0	4.84
10	15.04997	0.31	3.94	10.2	0.01	0.12	0.2	0.0	37.35	55.85	0.03	0.07	0.12	2.42	265.0	0.0	3.6	10.93	0.07	0.09	0.17	2.39	41.02	87.72	0.14	0.19	0.27	1.69	6.48
15	9.81804	0.87	5.38	12.55	0.01	0.11	0.19	7.35	57.58	99.44	0.04	0.08	0.13	2.19	265.0	0.05	6.49	16.07	0.1	0.15	0.25	6.99	48.49	110.02	0.18	0.24	0.31	4.17	8.89
20	7.233	1.22	5.66	12.09	0.01	0.13	0.2	6.71	62.1	141.34	0.06	0.1	0.15	2.28	265.0	1.17	7.61	21.51	0.13	0.18	0.25	6.58	72.96	159.84	0.23	0.31	0.43	5.0	11.1
25	5.86461	3.68	6.6	16.45	0.01	0.12	0.2	6.6	61.71	172.09	0.09	0.15	0.28	-1.17	265.03	-0.78	5.61	18.89	0.19	0.28	0.44	8.4	49.64	178.51	0.27	0.38	0.49	7.33	12.84
Mean:			**4.3**			**0.12**			**30.35**			**0.06**		**0.72**	**264.99**		**5.11**			**0.11**			**33.95**			**0.17**		**2.54**	**6.76**

Table A.38: Clustering details with Shuttle Control (normalized)

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	20	2000	245	0.2	0.12	2.1E+06	8	9.7E+05	264.96	0.0	2	1.7E+09	0.01	0.01	5	7.7E+05	0.04	0.0	3	3.9E+05	2.77	3.9E+06
3	20	2000	120	0.2	0.11	2.7E+06	10	1.7E+06	264.97	0.01	6	1.7E+09	0.01	0.02	8	1.8E+06	0.06	0.0	6	9.7E+05	2.98	2.5E+07
4	20	2000	150	0.2	0.11	4.4E+06	11	2.5E+06	264.97	0.01	3	1.7E+09	0.02	0.02	10	2.9E+06	0.08	0.0	6	1.5E+06	4.2	6.6E+07
5	20	2000	115	0.2	0.11	4.7E+06	13	3.8E+06	264.97	0.01	4	1.7E+09	0.03	0.03	9	3.5E+06	0.09	0.0	7	2.2E+06	4.84	7.5E+07
10	20	2000	84	0.2	0.12	8.6E+06	21	1.2E+07	264.98	0.02	6	1.7E+09	0.05	0.04	13	9.0E+06	0.19	0.0	10	5.6E+06	6.48	1.4E+08
15	20	2000	50	0.2	0.11	9.9E+06	22	1.9E+07	264.98	0.02	7	1.7E+09	0.08	0.07	14	1.5E+07	0.23	0.0	9	7.8E+06	8.89	1.9E+08
20	20	2000	32	0.2	0.13	1.1E+07	21	2.4E+07	264.97	0.03	8	1.7E+09	0.09	0.08	15	2.1E+07	0.31	0.0	10	1.2E+07	11.1	2.5E+08
25	20	2000	30	0.2	0.12	1.4E+07	28	4.0E+07	264.96	0.07	13	1.7E+09	0.14	0.13	17	2.9E+07	0.37	0.0	11	1.6E+07	12.84	3.1E+08

A.20 EEG Eye State

Table A.39: Summary of the results with EEG Eye State ($\times 10^8, m = 14980, n = 14$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	7845.09934	4.25	6.89	17.49	0.26	1.51	2.82	-0.0	0.87	17.49	0.0	0.0	0.01	-0.0	9.38	4.25	5.57	17.49	0.0	0.01	0.03	0.0	0.0	0.0	0.15	0.18	0.23	4.25	0.2
3	1833.88058	0.0	0.0	0.01	0.03	1.85	2.99	0.0	207.28	327.75	0.0	0.01	0.01	0.0	9.38	0.0	0.0	0.0	0.01	0.01	0.03	0.0	221.5	327.72	0.18	0.24	0.31	0.0	0.24
4	2.23605	0.0	0.0	0.0	0.29	1.65	2.91	0.0	215052.4	268821.85	0.0	0.01	0.01	0.0	9.38	0.0	0.0	0.0	0.01	0.01	0.01	0.02	241932.62	268832.07	0.25	0.3	0.35	0.0	0.39
5	1.33858	-0.0	13.46	29.91	0.03	1.44	2.93	29.91	224565.25	449118.74	0.01	0.01	0.02	-0.0	9.38	-0.0	16.45	29.91	0.01	0.01	0.03	30.0	381741.77	449125.07	0.36	0.43	0.48	-0.0	0.46
10	0.4531	-0.01	0.08	0.79	0.17	1.67	2.95	0.03	663475.87	1326896.85	0.01	0.02	0.05	0.81	9.38	0.01	0.22	0.81	0.03	0.04	0.07	189.5	331866.65	1326893.01	0.64	0.81	1.02	0.79	3.26
15	0.34653	0.0	0.86	2.6	0.18	1.81	2.95	0.64	1040923.59	1734970.3	0.01	0.03	0.06	0.47	9.41	0.08	0.87	1.95	0.05	0.08	0.18	6.32	694086.71	1734971.45	1.0	1.17	1.31	0.05	6.63
20	0.28986	0.03	0.99	3.16	0.2	1.82	2.98	0.47	829652.38	2074185.5	0.01	0.08	0.21	0.03	9.4	0.02	0.84	3.26	0.1	0.18	0.27	2.91	414879.54	2074154.31	1.26	1.46	1.62	0.0	10.57
25	0.25989	-0.05	0.6	2.28	0.12	1.56	2.98	0.6	1387955.4	2313366.1	0.01	0.06	0.15	0.34	9.43	0.13	0.62	2.17	0.13	0.18	0.34	1.97	1041022.25	2313347.19	1.63	1.8	2.04	0.16	16.49
Mean:			**2.86**			**1.66**			**545229.13**			**0.03**		**0.21**	**9.39**		**3.07**			**0.07**			**388218.88**			**0.8**		**0.66**	**4.78**

Table A.40: Clustering details with EEG Eye State

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	20	14979	721	3.0	1.51	4.3E+07	5	1.5E+05	9.37	0.0	2	1.1E+08	0.01	0.0	2	1.2E+05	0.18	0.0	4	1.3E+05	0.2	9.0E+06
3	20	14979	891	3.0	1.85	8.0E+07	7	3.0E+05	9.38	0.0	2	1.1E+08	0.01	0.0	2	1.9E+05	0.24	0.0	4	2.0E+05	0.24	9.2E+06
4	20	14979	811	3.0	1.65	9.7E+07	8	4.6E+05	9.37	0.0	2	1.1E+08	0.01	0.0	2	2.7E+05	0.3	0.0	6	3.7E+05	0.39	1.7E+07
5	20	14979	632	3.0	1.44	9.5E+07	14	1.1E+06	9.37	0.0	2	1.1E+08	0.01	0.0	6	6.6E+05	0.43	0.0	7	5.4E+05	0.46	2.0E+07
10	20	14979	535	3.0	1.67	1.6E+08	23	3.4E+06	9.38	0.01	6	1.1E+08	0.02	0.03	23	3.8E+06	0.8	0.0	15	2.3E+06	3.26	1.3E+08
15	20	14979	465	3.0	1.81	2.2E+08	18	4.1E+06	9.37	0.03	23	1.2E+08	0.03	0.05	25	6.3E+06	1.17	0.0	14	3.1E+06	6.63	2.8E+08
20	20	14979	339	3.0	1.82	2.1E+08	43	1.3E+07	9.38	0.03	16	1.2E+08	0.06	0.11	41	1.3E+07	1.45	0.01	23	6.8E+06	10.57	4.6E+08
25	20	14979	235	3.0	1.56	1.9E+08	24	9.1E+06	9.37	0.05	25	1.2E+08	0.07	0.11	38	1.5E+07	1.79	0.01	19	7.3E+06	16.49	7.4E+08

A.21 EEG Eye State (normalized)

Table A.41: Summary of the results with EEG Eye State (normalized) ($\times 10^1, m = 14980, n = 14$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t					E_A			t			E_A			t				
		min	mean	max	min	mean	max	min	mean	max	min	mean	max	E_A	t	min	mean	max	min	mean	max	min	mean	max	min	mean	max	E_A	t
2	1.15267	0.0	6.55	25.4	0.02	0.55	0.97	25.37	25.4	25.41	0.0	0.02	0.03	0.0	10.59	0.0	6.07	25.41	0.0	0.01	0.03	6.09	24.08	34.54	0.0	0.0	0.01	0.0	0.05
3	0.82423	0.0	9.34	35.98	0.02	0.58	1.0	68.96	69.03	69.07	0.01	0.01	0.02	0.0	10.59	0.0	10.48	41.24	0.01	0.01	0.03	45.68	76.99	85.55	0.0	0.0	0.01	0.01	0.12
4	0.5429	0.0	18.63	54.57	0.06	0.56	0.99	96.93	150.74	153.19	0.01	0.01	0.02	0.0	10.59	0.0	13.12	84.3	0.01	0.01	0.03	116.1	157.47	174.54	0.0	0.01	0.01	0.01	0.16
5	0.28952	0.0	31.14	81.23	0.04	0.56	0.9	265.87	363.78	367.31	0.01	0.01	0.02	0.0	10.59	0.0	35.31	148.44	0.01	0.02	0.04	277.88	375.07	405.37	0.0	0.01	0.02	0.0	0.22
10	0.10269	-0.01	0.51	1.11	0.13	0.58	1.0	556.29	806.35	912.38	0.01	0.03	0.06	-0.0	10.59	-0.0	0.75	1.16	0.03	0.04	0.06	897.05	1079.87	1241.74	0.01	0.01	0.01	0.22	0.85
15	0.07469	-0.07	0.48	2.45	0.11	0.57	0.97	369.72	867.29	1266.66	0.02	0.04	0.07	0.2	10.61	0.03	0.7	3.52	0.05	0.1	0.21	883.06	1484.63	1702.12	0.01	0.01	0.04	0.59	1.52
20	0.06125	-0.02	0.69	4.57	0.13	0.59	0.98	3.4	1016.71	1505.13	0.03	0.05	0.1	0.44	10.61	0.01	0.75	2.36	0.07	0.11	0.18	551.98	1542.54	2082.97	0.01	0.02	0.07	0.96	2.24
25	0.05385	-0.33	0.8	2.3	0.19	0.64	0.99	0.81	862.63	1198.77	0.03	0.08	0.2	0.67	10.65	-0.34	1.03	2.61	0.11	0.2	0.34	499.92	1682.9	2350.97	0.01	0.02	0.04	0.97	2.81
Mean:		**8.52**			**0.58**			**520.24**			**0.03**			**0.16**	**10.6**	**8.52**			**0.06**			**802.94**			**0.01**			**0.35**	**1.0**

Table A.42: Clustering details with EEG Eye State (normalized)

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	30	14979	253	1.0	0.55	1.5E+07	25	7.4E+05	10.59	0.0	2	1.1E+08	0.0	0.0	4	1.7E+05	0.0	0.0	2	7.3E+04	0.05	8.6E+05
3	30	14979	270	1.0	0.58	2.5E+07	14	6.3E+05	10.59	0.0	2	1.1E+08	0.01	0.0	5	3.4E+05	0.0	0.0	3	1.2E+05	0.12	2.0E+06
4	30	14979	248	1.0	0.56	3.0E+07	14	8.4E+05	10.59	0.0	2	1.1E+08	0.01	0.0	5	4.3E+05	0.0	0.0	3	2.0E+05	0.16	3.5E+06
5	30	14979	236	1.0	0.56	3.6E+07	17	1.3E+06	10.59	0.0	2	1.1E+08	0.01	0.01	10	9.6E+05	0.01	0.0	4	2.9E+05	0.22	5.9E+06
10	30	14979	162	1.0	0.58	5.2E+07	27	4.1E+06	10.59	0.01	7	1.1E+08	0.02	0.02	20	3.4E+06	0.01	0.0	5	6.9E+05	0.85	3.3E+07
15	30	14979	108	1.0	0.57	5.6E+07	27	6.1E+06	10.59	0.03	16	1.2E+08	0.04	0.07	28	7.0E+06	0.01	0.0	5	1.1E+06	1.52	5.9E+07
20	30	14979	90	1.0	0.59	6.5E+07	23	6.9E+06	10.58	0.03	11	1.2E+08	0.05	0.07	30	1.0E+07	0.02	0.0	6	1.8E+06	2.24	9.2E+07
25	30	14979	74	1.0	0.64	7.1E+07	31	1.1E+07	10.59	0.06	26	1.2E+08	0.07	0.13	38	1.5E+07	0.02	0.0	6	2.4E+06	2.81	1.2E+08

A.22 Pla85900

Table A.43: Summary of the results with Pla85900 ($\times 10^{15}, m = 85900, n = 2$)

k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t					E_A			t			E_A			t				
		min	mean	max	min	mean	max	min	mean	max	min	mean	max	E_A	t	min	mean	max	min	mean	max	min	mean	max	min	mean	max	E_A	t
2	3.74908	0.0	0.74	1.48	0.08	0.55	1.0	0.0	0.7	4.69	0.02	0.04	0.08	1.44	458.85	0.0	0.58	1.45	0.02	0.05	0.09	0.01	2.43	58.83	0.3	0.39	0.47	1.44	2.59
3	2.28057	0.0	0.04	0.14	0.02	0.48	0.97	0.0	0.06	0.09	0.02	0.07	0.14	0.08	458.86	0.0	0.05	0.09	0.02	0.07	0.14	0.04	3.04	66.91	0.42	0.53	0.69	0.0	4.14
5	1.33972	0.02	0.51	2.88	0.09	0.64	0.99	0.0	1.29	3.71	0.02	0.03	0.06	0.0	458.86	-0.0	1.19	3.71	0.03	0.05	0.1	0.04	4.21	70.41	0.6	0.81	0.96	2.77	6.6
10	0.68294	0.06	0.41	1.18	0.04	0.52	0.98	0.03	1.05	4.27	0.02	0.05	0.11	0.72	458.87	0.02	0.7	3.2	0.06	0.09	0.25	0.6	4.44	18.91	1.13	1.35	1.61	0.4	17.76
15	0.46029	0.1	0.55	2.56	0.04	0.51	0.98	0.08	0.99	2.7	0.03	0.07	0.17	0.76	458.89	0.04	1.0	2.51	0.09	0.13	0.22	0.47	4.21	16.58	1.73	2.12	2.72	0.92	23.87
20	0.34988	0.17	0.66	1.62	0.08	0.53	0.96	0.07	1.04	2.69	0.05	0.08	0.13	1.26	458.86	0.07	1.11	4.02	0.11	0.2	0.34	1.34	3.91	11.43	2.06	2.73	3.19	0.94	32.16
25	0.28259	0.15	0.92	1.68	0.1	0.58	1.01	0.04	1.1	2.44	0.05	0.1	0.19	0.52	458.89	-0.1	1.14	2.46	0.16	0.22	0.33	1.58	4.25	9.89	2.49	2.99	3.8	0.18	39.74
Mean:		**0.55**			**0.54**			**0.89**			**0.06**			**0.68**	**458.87**	**0.82**			**0.12**			**3.78**			**1.56**			**0.95**	**18.12**

Table A.44: Clustering details with Pla85900

k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	40	14000	215	1.0	0.55	2.0E+07	11	2.0E+06	458.84	0.01	5	3.7E+09	0.01	0.04	12	2.5E+06	0.39	0.0	7	1.2E+06	2.59	1.3E+08
3	40	14000	156	1.0	0.48	2.8E+07	23	5.9E+06	458.82	0.04	15	3.7E+09	0.01	0.05	18	5.2E+06	0.53	0.0	11	2.9E+06	4.14	4.0E+08
5	40	14000	243	1.0	0.64	7.9E+07	18	7.9E+06	458.82	0.04	25	3.7E+09	0.02	0.03	19	9.4E+06	0.81	0.0	15	6.3E+06	6.6	7.5E+08
10	40	14000	151	1.0	0.52	1.6E+08	38	3.2E+07	458.82	0.05	43	3.7E+09	0.04	0.06	36	3.3E+07	1.35	0.0	25	2.2E+07	17.76	-2.1E+09
15	40	14000	107	1.0	0.51	2.2E+08	49	6.3E+07	458.82	0.07	51	3.8E+09	0.06	0.07	38	5.3E+07	2.11	0.01	34	4.4E+07	23.87	-2.1E+09
20	40	14000	82	1.0	0.53	2.8E+08	52	8.9E+07	458.82	0.04	25	3.7E+09	0.09	0.11	42	7.7E+07	2.73	0.01	34	5.8E+07	32.16	-2.1E+09
25	40	14000	68	1.0	0.58	3.5E+08	54	1.2E+08	458.82	0.07	37	3.8E+09	0.1	0.12	42	9.7E+07	2.98	0.01	34	7.4E+07	39.74	-2.1E+09

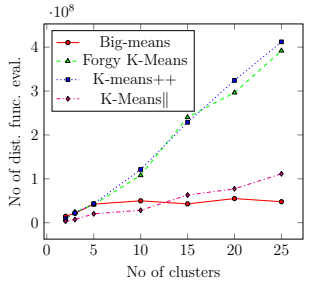
A.23 D15112

Table A.45: Summary of the results with D15112 ($\times 10^{11}, m = 15112, n = 2$)

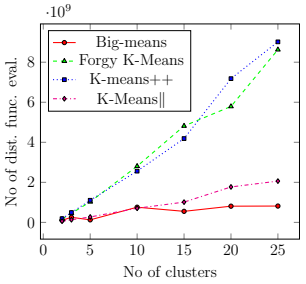
k	f^*	Big-means						Forgy K-Means						Ward's		K-Means++						K-Means‖						LMBM-Clust	
		E_A			t			E_A			t			E_A	t	E_A			t			E_A			t			E_A	t
		min	mean	max	min	mean	max	min	mean	max	min	mean	max			min	mean	max	min	mean	max	min	mean	max	min	mean	max		
2	3.68403	0.0	0.02	0.06	0.08	1.18	1.84	0.0	0.0	0.01	0.0	0.01	0.01	0.0	9.22	0.0	0.0	0.01	0.0	0.01	0.02	0.01	0.25	1.08	0.06	1.9	27.45	0.0	0.91
3	2.5324	0.0	0.04	0.09	0.1	0.88	1.81	0.02	0.04	0.04	0.01	0.02	0.03	0.04	9.22	0.01	0.04	0.05	0.01	0.02	0.04	0.12	3.63	12.74	0.05	0.06	0.07	-0.0	1.54
5	1.32707	0.01	2.26	16.73	0.07	0.76	1.65	0.0	1.11	16.64	0.01	0.01	0.02	0.0	9.22	0.0	0.9	13.48	0.01	0.01	0.03	0.04	7.93	17.11	0.07	0.09	0.1	-0.0	2.42
10	0.64491	0.08	1.33	3.92	0.16	1.27	1.94	0.0	2.69	21.12	0.01	0.01	0.03	0.04	9.21	0.01	1.43	4.18	0.02	0.02	0.03	0.71	5.39	13.99	0.14	0.17	0.21	1.41	3.64
15	0.43136	0.14	1.25	7.44	0.09	0.94	1.95	0.07	2.67	10.54	0.01	0.02	0.02	1.21	9.23	0.08	1.68	3.45	0.02	0.03	0.04	1.38	7.42	13.98	0.2	0.24	0.28	0.23	4.58
20	0.32177	0.36	1.25	2.7	0.66	1.4	1.88	0.79	2.48	8.53	0.01	0.02	0.04	1.85	9.22	0.29	2.03	4.4	0.03	0.04	0.07	0.76	6.51	15.35	0.27	0.34	0.4	0.24	5.42
25	0.25308	0.27	0.84	1.49	0.21	1.26	2.0	1.14	3.59	11.91	0.02	0.03	0.06	0.04	9.23	0.46	1.51	2.86	0.04	0.05	0.07	0.91	6.1	18.22	0.32	0.38	0.45	0.48	6.48
Mean:		**1.0**			**1.1**			**1.8**			**0.02**			**0.45**	**9.22**	**1.08**			**0.03**			**5.32**			**0.45**			**0.34**	**3.57**

Table A.46: Clustering details with D15112

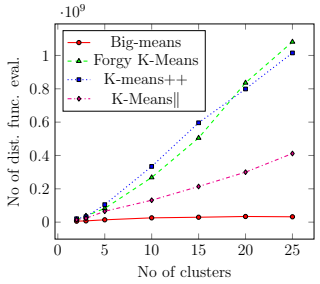
k	n_{exec}	Big-means					Forgy K-Means		Ward's				K-Means++				K-Means‖				LMBM-Clust	
		s	n_s	T	t	n_d	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{init}	t_{full}	n_{full}	n_d	t_{full}	n_d
2	15	8000	976	2.0	1.18	4.4E+07	9	2.8E+05	9.21	0.0	6	1.1E+08	0.0	0.01	8	3.0E+05	1.32	0.58	8	2.5E+05	0.91	1.9E+07
3	15	8000	637	2.0	0.88	5.5E+07	27	1.2E+06	9.21	0.01	15	1.1E+08	0.0	0.01	19	9.7E+05	0.06	0.0	12	5.3E+05	1.54	4.9E+07
5	15	8000	684	2.0	0.76	9.2E+07	21	1.6E+06	9.21	0.01	10	1.1E+08	0.01	0.01	16	1.4E+06	0.09	0.0	13	9.8E+05	2.42	8.9E+07
10	15	8000	741	2.0	1.27	3.1E+08	31	4.7E+06	9.21	0.0	11	1.2E+08	0.01	0.01	27	4.5E+06	0.17	0.0	21	3.1E+06	3.64	1.5E+08
15	15	8000	459	2.0	0.94	3.9E+08	36	8.2E+06	9.21	0.01	30	1.2E+08	0.02	0.01	29	7.2E+06	0.23	0.0	21	4.9E+06	4.58	2.0E+08
20	15	8000	542	2.0	1.4	7.1E+08	45	1.3E+07	9.21	0.01	12	1.2E+08	0.02	0.02	33	1.1E+07	0.34	0.0	27	8.3E+06	5.42	2.4E+08
25	15	8000	379	2.0	1.26	7.1E+08	53	2.0E+07	9.21	0.02	31	1.3E+08	0.03	0.03	46	1.9E+07	0.38	0.0	28	1.1E+07	6.48	2.9E+08



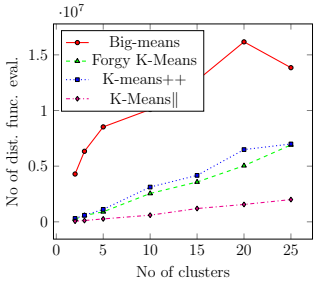
(a) CORD-19 Embeddings



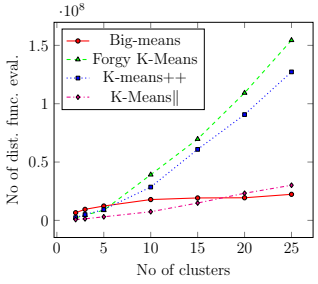
(b) HEPMASS



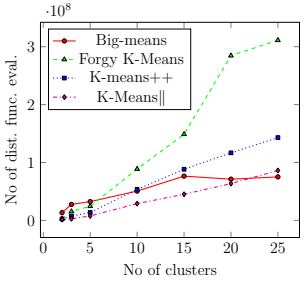
(c) US Census Data 1990



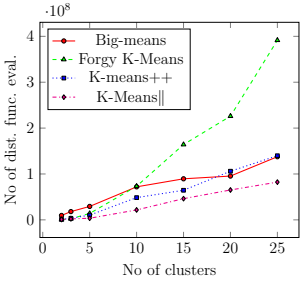
(d) Gisette



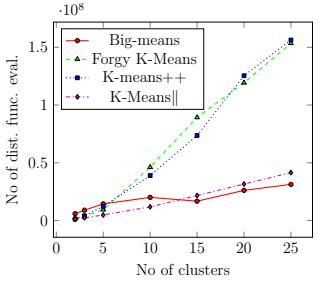
(e) Music Analysis



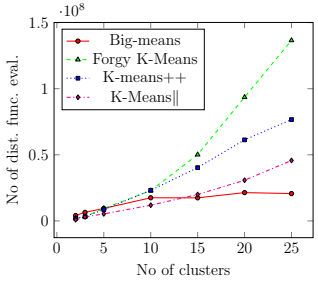
(f) Protein Homology



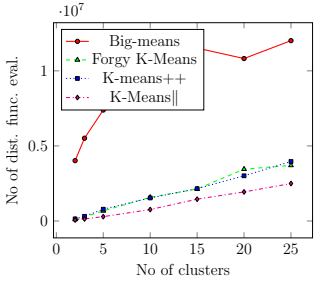
(g) MiniBooNE Particle Identification



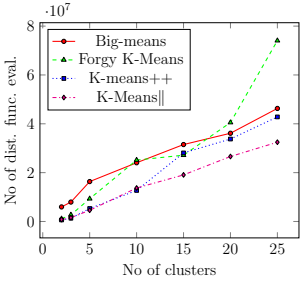
(h) MiniBooNE Particle Identification (normalized)



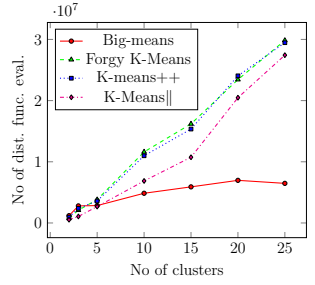
(i) MFCCs for Speech Emotion Recognition



(j) ISOLET

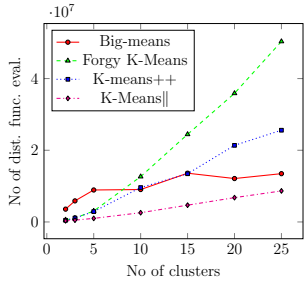


(k) Sensorless Drive Diagnosis

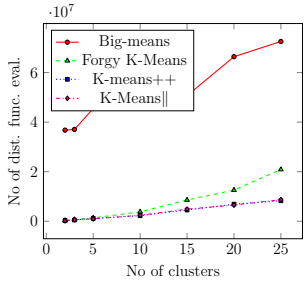


(l) Sensorless Drive Diagnosis (normalized)

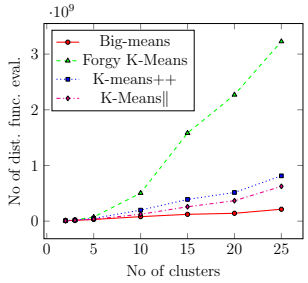
Figure A.1: Distance function evaluations. Set 1



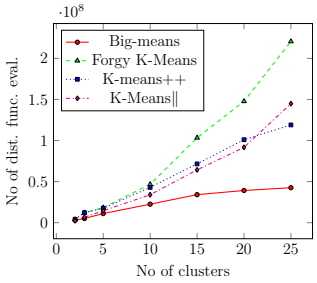
(a) Online News Popularity



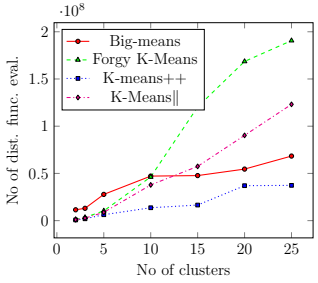
(b) Gas Sensor Array Drift



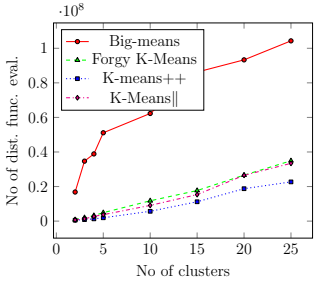
(c) 3D Road Network



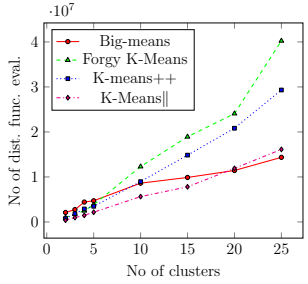
(d) Skin Segmentation



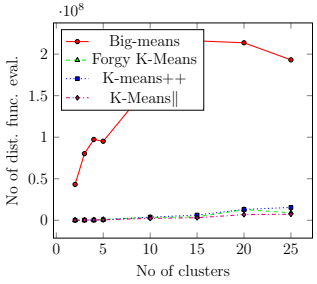
(e) KEGG Metabolic Relation Network (Directed)



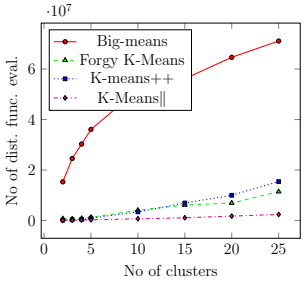
(f) Shuttle Control



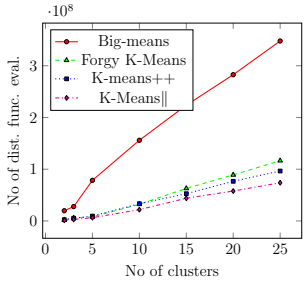
(g) Shuttle Control (normalized)



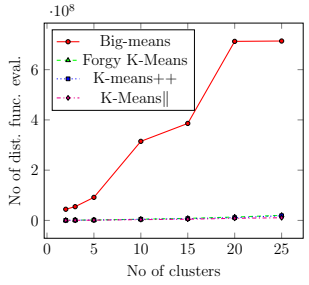
(h) EEG Eye State



(i) EEG Eye State (normalized)



(j) Pla85900



(k) D15112

Figure A.2: Distance function evaluations. Set 2

Appendix B

EXPERIMENTS WITH HPCLUST

B.1 CORD-19 Embeddings

Dimensions: $m = 599616$, $n = 768$.

Description: COVID-19 Open Research Dataset (CORD-19) is a resource of more than half a million scholarly articles about COVID-19, SARS-CoV-2, and related coronaviruses represended as embeddings in vectorized form.

Table B.1: Summary of the results with CORD-19 Embeddings ($\times 10^9$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	2.03893*	0.1082	0.006	0.001	15.834	0.0	15.834	6.423	0.006	0.001	15.225	3.105	20.295	9.046	0.006	0.0	15.556	15.461	27.333	14.943
3	1.9093*	0.10141	0.019	0.007	23.69	13.913	33.566	13.481	0.011	0.006	10.312	4.304	16.699	11.92	0.019	0.004	13.461	7.223	15.598	10.088
5	1.77676*	0.09433	0.145	0.066	22.952	10.562	27.398	11.366	0.015	0.003	11.529	2.162	12.459	7.228	0.015	0.002	21.706	4.474	27.337	5.387
10	1.62555*	0.08679	0.463	0.253	3.419	2.783	29.185	8.184	0.067	0.057	7.935	1.012	34.535	10.962	0.063	0.045	7.959	1.433	21.939	10.646
15	1.55295*	0.08276	0.282	0.104	12.552	7.758	31.399	8.814	0.073	0.124	19.717	5.888	34.974	7.468	0.128	0.061	14.53	1.741	20.63	8.791
20	1.49987*	0.07991	0.414	0.075	17.89	8.654	38.208	13.756	0.196	0.146	25.292	4.827	28.762	2.341	0.143	0.101	21.548	1.319	32.616	4.514
25	1.46394*	0.07789	0.166	0.195	16.347	7.16	31.45	9.114	0.092	0.07	32.139	5.446	36.157	4.77	0.153	0.097	32.654	4.919	33.721	6.338
Mean:			**0.213**		**16.098**		**29.577**		**0.066**		**17.45**		**26.269**		**0.075**		**18.202**		**25.596**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	2.03893*	0.1082	0.007	0.001	6.118	10.005	22.541	6.125	0.0	0.0	–	–	14.984	1.23	0.0	0.06	–	–	2.533	1.639
3	1.9093*	0.10141	0.011	0.005	7.245	4.692	18.988	8.346	0.011	1.406	–	–	45.337	10.538	0.058	0.028	–	–	9.368	4.529
5	1.77676*	0.09433	0.015	0.045	24.03	5.652	23.789	9.182	-0.002	0.234	–	–	104.197	13.797	2.161	1.416	–	–	15.807	1.399
10	1.62555*	0.08679	0.057	0.021	8.259	0.877	26.434	6.826	0.576	0.844	–	–	487.922	226.84	1.937	1.23	–	–	61.921	10.023
15	1.55295*	0.08276	0.111	0.153	15.05	2.097	27.231	7.537	0.342	0.229	–	–	887.432	1046.333	2.333	1.291	–	–	110.804	27.208
20	1.49987*	0.07991	0.161	0.184	25.027	3.976	38.075	7.841	0.233	0.319	–	–	1405.199	794.487	3.496	1.356	–	–	143.392	28.935
25	1.46394*	0.07789	0.18	0.079	26.508	3.56	32.961	7.091	0.056	0.217	–	–	1987.234	837.859	2.21	0.842	–	–	189.484	31.495
Mean:			**0.078**		**16.034**		**27.146**		**0.174**		–		**704.615**		**1.742**		–		**76.187**	

Table B.2: Clustering details with CORD-19 Embeddings

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	7	32000	82	40.0	3.7E+07	32000	434	40.0	1.6E+08	32000	569	40.0	1.5E+08	32000	511	37.333	2.667	1.6E+08	1.4E+07	1.3E+07
3	7	32000	147	40.0	5.4E+07	32000	257	40.0	2.0E+08	32000	233	40.0	2.0E+08	32000	297	32.0	8.0	2.0E+08	5.6E+07	4.9E+07
5	7	32000	104	40.0	8.0E+07	32000	129	40.0	2.5E+08	32000	321	40.0	2.6E+08	32000	267	21.333	18.667	2.6E+08	1.3E+08	1.0E+08
10	7	32000	75	40.0	1.2E+08	32000	215	40.0	3.5E+08	32000	123	40.0	3.4E+08	32000	147	24.0	16.0	3.3E+08	6.9E+08	4.2E+08
15	7	32000	41	40.0	1.4E+08	32000	110	40.0	3.7E+08	32000	35	40.0	3.6E+08	32000	73	26.667	13.333	3.5E+08	1.3E+09	7.9E+08
20	7	32000	46	40.0	1.7E+08	32000	37	40.0	3.8E+08	32000	54	40.0	3.8E+08	32000	45	8.0	32.0	3.3E+08	2.1E+09	1.0E+09
25	7	32000	32	40.0	1.9E+08	32000	30	40.0	3.7E+08	32000	23	40.0	3.7E+08	32000	25	32.0	8.0	3.4E+08	2.9E+09	1.5E+09

B.2 HEPMASS

Dimensions: $m = 10500000$, $n = 27$.

Description: The data set contains the 28 normalized features of physical particles that can be used for discovering the exotic ones in the field of high-energy physics.

Table B.3: Summary of the results with HEPMASS ($\times 10^8$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	2.48889*	0.01512	0.004	0.001	8.694	6.958	19.003	8.514	0.004	0.0	8.529	5.832	26.287	3.089	0.003	0.001	11.257	5.634	18.327	7.876
3	2.36789*	0.01439	0.009	0.62	16.367	2.089	19.541	6.92	0.005	0.003	7.582	3.877	13.209	6.614	0.008	0.436	4.764	3.203	16.956	8.51
5	2.21106*	0.01349	0.341	0.437	5.451	2.961	20.678	10.902	0.012	0.161	1.352	4.625	17.709	7.415	0.333	0.378	2.665	0.943	14.077	7.155
10	2.00353*	0.01223	0.289	0.078	5.034	6.332	18.004	8.518	0.086	0.069	2.335	0.651	16.864	10.671	0.122	0.066	1.619	0.369	13.306	5.406
15	1.89922*	0.01157	0.397	0.191	5.965	0.0	12.862	7.789	0.094	0.068	6.256	6.402	23.113	5.153	0.155	0.12	4.364	1.089	16.703	6.439
20	1.82904*	0.01114	0.322	0.051	15.461	4.451	17.015	8.134	0.156	0.087	6.688	5.447	20.223	7.415	0.209	0.089	3.768	5.851	20.596	4.425
25	1.77524*	0.01082	0.189	0.179	3.61	4.257	22.461	6.812	0.111	0.032	3.925	1.402	24.055	4.933	0.279	0.151	5.554	4.163	19.474	7.38
Mean:			**0.222**		**8.655**		**18.509**		**0.067**		**5.238**		**20.209**		**0.159**		**4.856**		**17.063**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	2.48889*	0.01512	0.003	0.001	19.63	6.539	21.339	7.536	0.0	0.0	–	–	14.412	1.882	0.0	0.0	–	–	3.128	0.266
3	2.36789*	0.01439	0.005	0.001	9.107	4.481	18.658	7.2	0.0	0.436	–	–	30.808	15.829	0.347	1.087	–	–	5.228	0.278
5	2.21106*	0.01349	0.008	0.161	2.253	4.243	15.988	8.928	0.323	0.114	–	–	79.275	8.253	0.985	0.993	–	–	8.819	0.499
10	2.00353*	0.01223	0.112	0.072	1.658	0.439	23.171	6.864	0.217	0.257	–	–	398.39	138.675	2.767	1.451	–	–	28.289	1.781
15	1.89922*	0.01157	0.115	0.163	3.747	5.177	18.969	7.195	0.289	0.188	–	–	706.048	332.25	1.634	1.369	–	–	50.802	3.377
20	1.82904*	0.01114	0.12	0.101	4.547	9.065	16.949	9.017	0.121	0.166	–	–	1103.684	294.319	2.351	0.863	–	–	67.597	2.791
25	1.77524*	0.01082	0.186	0.139	5.863	4.647	24.828	7.007	0.344	0.269	–	–	1229.349	342.28	2.092	0.584	–	–	85.084	5.815
Mean:			**0.078**		**6.686**		**19.986**		**0.185**		–		**508.852**		**1.454**		–		**35.564**	

Table B.4: Clustering details with HEPMASS

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	7	64000	25	30.0	3.2E+07	64000	256	30.0	1.1E+08	64000	172	30.0	1.0E+08	64000	188	17.0	13.0	1.0E+08	5.7E+08	4.0E+08
3	7	64000	27	30.0	5.6E+07	64000	116	30.0	2.0E+08	64000	155	30.0	1.9E+08	64000	164	7.0	23.0	2.0E+08	1.4E+09	1.1E+09
5	7	64000	26	30.0	9.6E+07	64000	166	30.0	3.8E+08	64000	121	30.0	3.6E+08	64000	135	7.0	23.0	3.7E+08	4.3E+09	2.7E+09
10	7	64000	23	30.0	2.0E+08	64000	135	30.0	8.0E+08	64000	112	30.0	7.4E+08	64000	189	16.0	14.0	7.7E+08	2.4E+10	1.1E+10
15	7	64000	16	30.0	3.1E+08	64000	189	30.0	1.3E+09	64000	142	30.0	1.2E+09	64000	148	9.0	21.0	1.3E+09	4.4E+10	2.2E+10
20	7	64000	21	30.0	4.4E+08	64000	154	30.0	1.9E+09	64000	162	30.0	1.8E+09	64000	129	28.0	2.0	1.8E+09	7.0E+10	2.9E+10
25	7	64000	26	30.0	5.7E+08	64000	175	30.0	2.4E+09	64000	140	30.0	2.2E+09	64000	170	22.0	8.0	2.3E+09	7.8E+10	3.7E+10

B.3 US Census Data 1990

Dimensions: $m = 2458285$, $n = 68$.

Description: The data set was obtained from the (U.S. Department of Commerce) Census Bureau website and contains a one percent sample of the Public Use Microdata Samples (PUMS) person records drawn from the entire 1990 U.S. census sample.

Table B.5: Summary of the results with US Census Data 1990 ($\times 10^8$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	18.39812*	0.04235	0.107	0.107	1.569	0.665	1.547	0.737	0.278	0.191	1.058	0.722	2.102	0.673	0.233	0.174	0.983	0.685	1.825	0.68
3	6.1591*	0.01444	0.069	66.222	0.813	1.121	1.191	0.959	0.078	0.03	1.292	0.691	1.852	0.859	0.074	0.023	0.856	0.741	1.376	0.951
5	3.35214*	0.00827	2.179	9.495	0.142	0.271	1.937	0.929	0.105	0.044	0.112	0.035	1.966	0.753	0.13	1.488	0.108	0.319	1.343	0.708
10	2.36352*	0.00599	4.682	2.985	0.166	0.588	1.926	0.867	2.413	1.546	0.179	0.137	2.063	0.961	3.296	2.01	0.172	0.039	2.068	0.86
15	2.04097*	0.00508	4.538	4.329	0.368	0.71	1.774	0.718	2.141	1.11	0.258	0.164	1.769	0.75	1.829	1.228	0.246	0.074	1.957	0.719
20	1.81278*	0.00446	5.921	3.048	0.89	0.499	1.407	0.672	2.17	0.794	0.358	0.347	2.132	0.935	2.858	1.073	0.455	0.351	1.755	0.609
25	1.64602*	0.00408	4.439	1.401	0.487	0.586	1.613	0.952	3.178	0.842	0.457	0.381	1.806	0.875	3.034	0.946	0.434	0.188	1.928	0.73
Mean:			**3.134**		**0.634**		**1.628**		**1.48**		**0.531**		**1.956**		**1.636**		**0.465**		**1.75**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	18.39812*	0.04235	0.193	0.101	0.529	0.722	2.371	0.787	0.0	0.0	–	–	0.824	0.235	0.0	0.0	–	–	0.334	0.019
3	6.1591*	0.01444	0.083	0.027	1.114	0.72	1.452	0.882	162.972	61.576	–	–	1.837	1.254	170.038	55.912	–	–	0.652	0.036
5	3.35214*	0.00827	0.117	0.037	0.106	0.037	1.563	0.7	356.299	149.498	–	–	16.932	7.178	216.673	185.79	–	–	1.675	0.106
10	2.36352*	0.00599	3.426	1.645	0.168	0.042	2.665	1.017	12.78	250.618	–	–	41.385	15.171	21.403	168.656	–	–	4.014	0.287
15	2.04097*	0.00508	2.063	1.189	0.279	0.159	2.463	0.757	9.039	185.544	–	–	74.563	18.755	17.079	191.017	–	–	6.282	0.398
20	1.81278*	0.00446	3.548	1.232	0.572	0.434	1.873	0.766	12.895	6.605	–	–	117.544	73.095	16.362	10.313	–	–	9.041	0.82
25	1.64602*	0.00408	2.641	1.094	0.41	0.262	2.147	0.78	11.136	5.757	–	–	179.535	60.011	15.337	6.458	–	–	11.032	0.508
Mean:			**1.724**		**0.454**		**2.077**		**80.732**		–		**61.803**		**65.27**		–		**4.719**	

Table B.6: Clustering details with US Census Data 1990

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	20	6000	19	3.0	5.9E+06	6000	162	3.0	1.1E+07	6000	120	3.0	1.1E+07	6000	170	0.2	2.8	1.1E+07	1.5E+07	1.9E+07
3	20	6000	14	3.0	8.7E+06	6000	136	3.0	1.7E+07	6000	106	3.0	1.7E+07	6000	100	2.1	0.9	1.7E+07	3.7E+07	4.4E+07
5	20	6000	18	3.0	1.5E+07	6000	143	3.0	3.2E+07	6000	97	3.0	2.9E+07	6000	113	0.6	2.4	3.0E+07	3.6E+08	1.5E+08
10	20	6000	20	3.0	3.5E+07	6000	120	3.0	7.5E+07	6000	132	3.0	7.0E+07	6000	176	2.4	0.6	7.9E+07	9.5E+08	5.1E+08
15	20	6000	23	3.0	5.5E+07	6000	88	3.0	1.2E+08	6000	104	3.0	1.2E+08	6000	128	1.9	1.1	1.2E+08	1.7E+09	9.0E+08
20	20	6000	16	3.0	7.8E+07	6000	92	3.0	1.7E+08	6000	66	3.0	1.5E+08	6000	78	0.1	2.9	1.5E+08	2.7E+09	1.3E+09
25	20	6000	12	3.0	9.4E+07	6000	60	3.0	2.0E+08	6000	64	3.0	2.0E+08	6000	72	2.5	0.5	2.0E+08	4.1E+09	1.7E+09

B.4 Gisette

Dimensions: $m = 13500$, $n = 5000$.

Description: patterns for handwritten digit recognition problem.

Table B.7: Summary of the results with Gisette ($\times 10^{12}$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	4.19944	3.1048	0.009	0.005	2.673	1.476	3.135	1.445	0.007	0.006	3.941	0.683	4.994	0.75	0.009	0.005	4.327	0.746	4.623	0.759
3	4.11596	3.04579	0.036	0.148	1.315	1.4	2.646	1.244	0.029	0.013	5.224	0.563	5.991	0.964	0.023	0.017	5.001	0.426	5.271	0.749
5	4.02303	2.97834	0.077	0.03	3.049	0.836	3.353	0.894	0.064	0.037	8.468	0.901	8.529	0.875	0.081	0.037	8.397	1.456	8.484	1.568
10	3.87672	2.87532	0.165	0.099	4.147	0.874	4.501	0.82	0.156	0.045	16.714	0.814	18.757	2.217	0.132	0.061	16.53	1.032	18.299	1.802
15	3.81766	2.81586	-0.282	0.051	5.514	0.535	5.47	0.646	-0.297	0.048	25.545	3.208	26.532	5.26	-0.315	0.048	25.196	2.045	28.983	2.846
20	3.81436	2.77677	-1.6	0.048	6.593	0.908	6.593	0.713	-1.628	0.045	32.543	3.114	38.244	4.044	-1.627	0.066	32.486	1.482	34.75	2.311
25	3.74937	2.74501	-1.002	0.072	7.495	0.988	6.831	1.347	-1.022	0.053	43.202	3.331	46.72	3.559	-1.055	0.055	40.976	4.052	45.528	3.191
Mean:			**-0.371**		**4.398**		**4.647**		**-0.384**		**19.377**		**21.395**		**-0.393**		**18.988**		**20.848**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	4.19944	3.1048	0.008	0.004	4.459	0.432	4.499	0.702	0.0	0.0	–	–	3.691	1.633	0.006	0.003	–	–	3.726	1.139
3	4.11596	3.04579	0.023	0.016	5.623	1.684	5.777	1.243	0.0	0.0	–	–	7.901	2.493	0.01	0.002	–	–	8.518	2.308
5	4.02303	2.97834	0.071	0.032	8.021	0.722	8.617	1.411	0.011	0.027	–	–	32.88	20.091	0.037	0.041	–	–	16.183	8.116
10	3.87672	2.87532	0.127	0.046	16.899	1.893	19.183	2.955	0.038	0.049	–	–	45.665	21.177	0.116	0.042	–	–	31.801	17.762
15	3.81766	2.81586	-0.301	0.026	25.974	2.107	27.735	2.35	-0.442	0.046	–	–	59.415	23.158	-0.332	0.069	–	–	43.361	10.205
20	3.81436	2.77677	-1.658	0.044	32.582	2.488	36.983	4.158	-1.782	0.045	–	–	106.076	27.759	-1.69	0.058	–	–	60.231	25.017
25	3.74937	2.74501	-1.049	0.042	41.765	2.526	45.036	3.52	-1.215	0.082	–	–	114.912	27.943	-1.11	0.079	–	–	68.469	23.838
Mean:			**-0.397**		**19.332**		**21.119**		**-0.484**		–		**52.934**		**-0.423**		–		**33.184**	

Table B.8: Clustering details with Gisette

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	15	10000	8	5.0	1.1E+06	10000	20	5.0	3.4E+06	10000	18	5.0	3.4E+06	10000	15	4.5	0.5	3.3E+06	7.0E+05	6.7E+05
3	15	10000	5	5.0	1.5E+06	10000	7	5.0	4.0E+06	10000	4	5.0	4.1E+06	10000	5	1.833	3.167	3.9E+06	1.6E+06	1.6E+06
5	15	10000	5	5.0	2.0E+06	10000	4	5.0	6.3E+06	10000	6	5.0	6.4E+06	10000	5	3.167	1.833	6.4E+06	6.8E+06	3.2E+06
10	15	10000	1	5.0	2.9E+06	10000	6	5.0	1.6E+07	10000	6	5.0	1.6E+07	10000	5	3.833	1.167	1.7E+07	9.7E+06	6.8E+06
15	15	10000	1	5.0	3.9E+06	10000	5	5.0	2.5E+07	10000	7	5.0	2.5E+07	10000	6	3.0	2.0	2.4E+07	1.3E+07	9.5E+06
20	15	10000	1	5.0	4.4E+06	10000	6	5.0	3.5E+07	10000	3	5.0	3.3E+07	10000	6	2.333	2.667	3.5E+07	2.3E+07	1.3E+07
25	15	10000	1	5.0	6.1E+06	10000	6	5.0	4.4E+07	10000	6	5.0	4.4E+07	10000	5	4.667	0.333	4.4E+07	2.5E+07	1.5E+07

B.5 Music Analysis

Dimensions: $m = 106574$, $n = 518$.

Description: a dataset for music analysis which contains different spectral and statistical attributes for each music track.

Table B.9: Summary of the results with Music Analysis ($\times 10^{11}$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	5.00474*	0.26351	0.065	7.815	4.602	2.312	4.495	2.16	0.097	0.054	1.824	1.797	4.255	2.473	0.073	0.025	1.266	1.203	5.064	2.03
3	3.83748*	0.20356	0.076	0.037	2.503	2.658	4.018	2.625	0.163	0.077	4.177	1.92	4.841	1.995	0.132	0.047	2.619	1.984	4.483	2.256
5	2.74249*	0.14584	0.274	1.459	1.709	2.075	4.109	2.177	0.195	0.137	2.379	2.027	4.536	2.182	0.21	1.192	1.539	0.946	4.358	1.809
10	1.87296*	0.10086	1.911	0.823	2.045	2.229	4.75	2.346	0.51	0.645	1.938	1.989	4.025	2.093	0.658	0.766	2.446	1.825	4.716	1.959
15	1.54422*	0.08235	1.181	0.352	5.033	2.527	5.469	1.896	1.002	0.506	5.801	2.243	6.335	2.091	1.104	0.615	5.006	1.346	5.922	1.963
20	1.35315*	0.07212	1.416	0.683	2.412	2.113	4.326	2.584	1.287	0.5	6.013	1.922	5.91	2.243	1.398	0.84	4.482	1.778	5.824	2.016
25	1.22622*	0.06535	1.466	0.814	4.223	1.683	4.973	1.925	1.912	0.483	5.984	1.433	6.462	1.676	2.224	0.697	5.937	1.84	6.796	1.793
Mean:			**0.913**		**3.218**		**4.591**		**0.738**		**4.017**		**5.195**		**0.829**		**3.328**		**5.309**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	5.00474*	0.26351	0.081	0.042	2.83	1.904	3.613	2.307	-0.0	9.283	–	–	2.863	0.593	1.266	6.592	–	–	0.446	0.06
3	3.83748*	0.20356	0.142	0.048	3.071	1.871	4.941	1.695	-0.0	4.505	–	–	4.797	2.883	2.19	13.482	–	–	1.174	0.314
5	2.74249*	0.14584	0.208	0.494	2.046	1.981	5.308	2.497	-0.001	1.834	–	–	11.907	2.354	1.223	29.104	–	–	2.204	0.38
10	1.87296*	0.10086	0.604	0.786	2.652	1.685	5.809	2.564	1.448	1.175	–	–	56.994	26.9	9.839	8.146	–	–	6.313	0.994
15	1.54422*	0.08235	1.565	0.612	4.07	2.336	6.531	2.011	0.649	0.425	–	–	138.407	34.68	5.648	4.717	–	–	10.277	1.354
20	1.35315*	0.07212	1.405	0.658	5.708	2.343	6.508	2.543	0.597	0.594	–	–	151.525	44.902	7.064	3.501	–	–	13.586	2.985
25	1.22622*	0.06535	1.945	0.795	6.795	2.159	6.584	2.231	0.611	0.548	–	–	239.709	65.327	6.692	4.415	–	–	17.212	2.663
Mean:			**0.85**		**3.882**		**5.613**		**0.472**		–		**86.6**		**4.846**		–		**7.316**	

Table B.10: Clustering details with Music Analysis

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	20	6000	278	8.0	2.0E+07	6000	964	8.0	8.7E+07	6000	1249	8.0	8.6E+07	6000	834	1.333	6.667	8.5E+07	4.9E+06	3.3E+06
3	20	6000	157	8.0	2.7E+07	6000	644	8.0	9.7E+07	6000	629	8.0	9.7E+07	6000	680	1.6	6.4	9.6E+07	9.1E+06	8.0E+06
5	20	6000	116	8.0	3.8E+07	6000	318	8.0	1.1E+08	6000	282	8.0	1.1E+08	6000	345	1.333	6.667	1.1E+08	2.4E+07	1.7E+07
10	20	6000	50	8.0	5.5E+07	6000	59	8.0	1.2E+08	6000	84	8.0	1.2E+08	6000	99	6.133	1.867	1.1E+08	1.2E+08	6.3E+07
15	20	6000	34	8.0	6.0E+07	6000	59	8.0	1.2E+08	6000	46	8.0	1.2E+08	6000	46	4.533	3.467	1.1E+08	3.0E+08	1.0E+08
20	20	6000	14	8.0	6.4E+07	6000	29	8.0	1.2E+08	6000	30	8.0	1.2E+08	6000	21	0.533	7.467	1.0E+08	3.3E+08	1.5E+08
25	20	6000	16	8.0	6.6E+07	6000	21	8.0	1.2E+08	6000	23	8.0	1.2E+08	6000	10	0.267	7.733	8.8E+07	5.2E+08	1.9E+08

B.6 Protein Homology

Dimensions: $m = 145751$, $n = 74$.

Description: a data set for protein homology prediction which contains a features describing the match (e.g. the score of a sequence alignment) between the native protein sequence and the sequence that is tested for homology.

Table B.11: Summary of the results with Protein Homology ($\times 10^{11}$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	15.20433*	4.88318	1.848	0.686	2.345	0.497	2.272	0.746	1.878	0.029	0.873	0.732	1.514	0.949	1.866	0.663	1.5	0.721	2.351	0.583
3	8.07129*	2.89651	0.521	0.553	2.183	0.98	1.949	0.908	0.851	0.609	1.203	0.666	2.23	1.017	0.621	0.457	1.155	0.866	2.187	0.952
5	5.30537*	1.86379	0.804	0.622	1.397	0.999	1.554	0.888	0.784	0.838	1.432	0.799	1.569	0.916	0.651	0.424	1.105	0.391	1.401	0.649
10	3.3767*	1.26637	0.198	0.787	1.784	0.887	2.311	0.915	0.244	0.21	2.264	0.808	2.69	0.521	0.235	0.21	2.417	0.506	2.718	0.502
15	2.86473*	1.08655	1.166	0.849	1.665	0.798	2.274	0.927	0.905	0.429	3.293	0.636	3.547	0.727	0.733	0.468	3.288	0.775	3.379	0.777
20	2.5732*	0.98195	0.782	0.531	1.535	0.899	2.782	0.896	0.761	0.414	3.874	0.656	4.003	0.823	1.012	0.495	3.692	0.6	4.596	1.048
25	2.38539*	0.90731	1.035	0.745	1.045	0.398	2.556	0.886	0.719	0.736	4.189	0.826	4.557	1.009	1.22	0.664	4.481	1.436	4.775	1.223
Mean:			**0.908**		**1.708**		**2.242**		**0.878**		**2.447**		**2.873**		**0.906**		**2.52**		**3.058**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	15.20433*	4.88318	1.874	0.025	0.995	1.107	2.419	0.768	1.824	0.0	–	–	0.44	0.048	1.825	0.004	–	–	0.307	0.037
3	8.07129*	2.89651	0.811	0.617	2.182	1.037	2.114	1.027	0.0	0.0	–	–	1.241	0.216	0.017	53.129	–	–	0.899	0.154
5	5.30537*	1.86379	1.689	0.508	1.242	0.849	2.808	0.666	0.001	0.0	–	–	3.087	0.22	15.471	10.927	–	–	1.797	0.297
10	3.3767*	1.26637	0.387	0.211	2.967	0.52	3.022	0.539	18.12	0.0	–	–	12.253	2.843	25.368	10.08	–	–	5.637	2.004
15	2.86473*	1.08655	0.949	0.462	3.528	1.02	3.837	1.13	23.941	0.064	–	–	31.292	7.263	31.217	10.86	–	–	11.102	1.696
20	2.5732*	0.98195	0.623	0.542	4.412	0.955	4.681	1.218	28.605	0.245	–	–	35.658	11.432	33.088	11.515	–	–	15.362	3.452
25	2.38539*	0.90731	0.982	0.808	5.021	0.448	4.72	1.107	31.848	0.149	–	–	51.196	12.361	37.4	7.777	–	–	20.165	4.787
Mean:			**1.045**		**2.907**		**3.371**		**14.906**		**–**		**19.31**		**20.627**		**–**		**7.896**	

Table B.12: Clustering details with Protein Homology

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	15	56000	79	3.5	4.1E+07	56000	259	3.5	2.1E+08	56000	341	3.5	2.2E+08	56000	428	3.267	0.233	2.2E+08	6.7E+06	5.6E+06
3	15	56000	70	3.5	6.8E+07	56000	244	3.5	2.6E+08	56000	272	3.5	2.7E+08	56000	259	2.567	0.933	2.6E+08	2.0E+07	1.6E+07
5	15	56000	45	3.5	9.1E+07	56000	103	3.5	3.0E+08	56000	57	3.5	3.0E+08	56000	186	0.933	2.567	2.8E+08	5.8E+07	3.9E+07
10	15	56000	27	3.5	1.4E+08	56000	38	3.5	3.3E+08	56000	28	3.5	3.1E+08	56000	13	0.233	3.267	2.0E+08	2.5E+08	1.7E+08
15	15	56000	11	3.5	1.8E+08	56000	14	3.5	3.6E+08	56000	10	3.5	3.4E+08	56000	6	0.233	3.267	3.1E+08	6.5E+08	3.6E+08
20	15	56000	15	3.5	1.9E+08	56000	5	3.5	3.8E+08	56000	6	3.5	4.0E+08	56000	6	1.167	2.333	3.7E+08	7.5E+08	4.9E+08
25	15	56000	10	3.5	2.0E+08	56000	3	3.5	4.5E+08	56000	6	3.5	4.5E+08	56000	4	3.15	0.35	4.3E+08	1.0E+09	7.0E+08

B.7 MiniBooNE Particle Identification

Dimensions: $m = 130064$, $n = 50$.

Description: a data set for distinguishing electron neutrinos (signal) from muon neutrinos (background) which contains different particle variables for each event.

Table B.13: Summary of the results with MiniBooNE Particle Identification ($\times 10^{10}$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	8.92236	8.90824	0.0	0.0	0.961	0.493	1.719	0.593	0.0	0.0	0.546	0.757	1.931	0.836	0.0	0.0	0.624	0.462	1.507	0.735
3	5.22601	5.2178	0.0	5.409	0.881	0.86	1.598	0.83	0.0	0.0	0.497	0.317	1.844	0.773	0.0	0.001	0.559	0.448	2.014	0.656
5	1.82252	1.82055	0.005	29.133	0.542	1.107	1.646	1.114	0.005	0.006	1.349	0.526	1.784	0.63	0.003	0.003	1.362	0.464	2.068	0.43
10	0.9092	0.90911	0.094	702427.909	1.583	0.613	2.219	0.65	0.033	0.043	2.63	0.329	3.05	0.945	0.051	0.031	2.705	0.412	3.065	0.461
15	0.63506	0.64964	2.395	1.575	1.162	0.483	2.284	0.567	0.173	0.667	3.973	0.442	4.099	0.67	0.141	0.391	3.87	0.793	4.345	1.781
20	0.50863	0.54514	1.034	3.267	1.003	0.285	2.073	0.444	0.085	1255623.008	4.681	0.536	6.863	1.949	0.214	0.291	4.608	0.827	6.948	1.716
25	0.44425	0.44476	0.026	2.267	1.208	0.698	2.961	0.565	-0.303	0.133	7.802	1.262	8.719	1.595	-0.391	0.113	6.073	0.774	7.816	2.224
Mean:			**0.508**		**1.048**		**2.071**		**-0.001**		**3.068**		**4.041**		**0.003**		**2.829**		**3.966**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	8.92236	8.90824	0.0	0.0	0.453	0.468	1.686	0.803	0.0	140555.682	–	–	0.135	0.109	286908.084	140555.682	–	–	0.197	0.069
3	5.22601	5.2178	0.0	0.0	0.817	0.383	1.932	0.66	0.0	166530.775	–	–	0.416	0.236	0.0	122199.794	–	–	0.343	0.125
5	1.82252	1.82055	0.008	0.005	2.417	0.64	2.724	0.639	116.777	55.608	–	–	1.563	0.473	116.777	57.841	–	–	1.636	0.439
10	0.9092	0.90911	0.03	0.053	2.787	1.204	4.616	0.816	0.002	0.0	–	–	13.383	3.122	0.002	0.001	–	–	12.268	2.158
15	0.63506	0.64964	0.123	0.048	3.514	0.685	5.845	1.681	3.883	0.764	–	–	17.098	4.58	3.883	0.777	–	–	15.296	4.864
20	0.50863	0.54514	0.091	0.169	5.003	0.552	5.87	1.558	7.051	0.439	–	–	24.047	4.968	7.052	0.603	–	–	24.811	6.02
25	0.44425	0.44476	-0.267	0.182	6.377	0.932	7.623	1.661	8.936	0.202	–	–	29.885	4.702	8.936	0.38	–	–	28.882	7.385
Mean:			**-0.002**		**3.053**		**4.328**		**19.521**		–		**12.361**		**41006.39**		–		**11.919**	

Table B.14: Clustering details with MiniBooNE Particle Identification

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	15	130000	59	3.0	5.5E+07	130000	201	3.0	2.1E+08	130000	182	3.0	2.1E+08	130000	203	2.5	0.5	1.9E+08	2.3E+06	4.7E+06
3	15	130000	42	3.0	7.2E+07	130000	161	3.0	2.5E+08	130000	173	3.0	2.6E+08	130000	159	2.5	0.5	2.5E+08	8.6E+06	8.2E+06
5	15	130000	39	3.0	1.1E+08	130000	74	3.0	3.1E+08	130000	98	3.0	3.1E+08	130000	74	0.2	2.8	2.2E+08	4.5E+07	4.8E+07
10	15	130000	29	3.0	1.7E+08	130000	8	3.0	3.8E+08	130000	12	3.0	3.7E+08	130000	11	2.0	1.0	3.4E+08	4.1E+08	3.8E+08
15	15	130000	21	3.0	2.1E+08	130000	4	3.0	5.8E+08	130000	4	3.0	5.9E+08	130000	6	0.5	2.5	6.2E+08	5.4E+08	4.9E+08
20	15	130000	14	3.0	2.3E+08	130000	5	3.0	8.2E+08	130000	5	3.0	8.5E+08	130000	4	2.9	0.1	8.5E+08	7.6E+08	7.8E+08
25	15	130000	16	3.0	2.5E+08	130000	5	3.0	1.1E+09	130000	5	3.0	9.9E+08	130000	3	1.7	1.3	1.1E+09	9.5E+08	9.3E+08

B.8 MiniBooNE Particle Identification (normalized)

Dimensions: $m = 130064$, $n = 50$.

Description: a data set for distinguishing electron neutrinos (signal) from muon neutrinos (background) which contains different particle variables for each event. Min-max scaling was used for normalization of data set values for better clusterization.

Table B.15: Summary of the results with MiniBooNE Particle Identification (normalized) ($\times 10^2$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	28.01938*	2.49407	0.014	150.663	0.37	0.279	0.417	0.303	0.019	0.01	0.142	0.209	0.546	0.302	0.027	0.009	0.219	0.156	0.625	0.321
3	19.85673*	1.75033	0.031	3.034	0.352	0.262	0.492	0.231	0.026	0.014	0.15	0.168	0.534	0.293	0.031	0.014	0.152	0.135	0.52	0.303
5	12.10267*	1.11597	0.12	1.745	0.023	0.013	0.647	0.301	0.089	0.028	0.066	0.016	0.604	0.299	0.087	0.043	0.064	0.027	0.459	0.249
10	8.57382*	0.76679	0.668	0.528	0.612	0.378	0.479	0.322	0.471	0.33	0.444	0.196	0.627	0.263	0.647	0.564	0.692	0.202	0.837	0.209
15	7.24131*	0.64941	0.619	0.26	0.27	0.294	0.467	0.26	0.75	0.287	0.55	0.184	0.76	0.222	0.772	0.445	0.66	0.197	0.763	0.221
20	6.30493*	0.56979	1.164	0.703	0.463	0.253	0.586	0.298	1.282	0.747	1.0	0.273	1.045	0.208	0.963	0.577	0.848	0.209	0.951	0.222
25	5.71335*	0.51724	1.147	0.47	0.693	0.269	0.738	0.253	1.209	0.447	1.108	0.231	1.171	0.232	1.363	0.605	0.915	0.219	1.157	0.303
Mean:			**0.538**		**0.398**		**0.546**		**0.549**		**0.494**		**0.755**		**0.556**		**0.507**		**0.759**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	28.01938*	2.49407	0.02	0.009	0.239	0.194	0.582	0.231	0.0	336.097	–	–	0.245	0.099	2.67	1.876	–	–	0.064	0.019
3	19.85673*	1.75033	0.03	0.014	0.222	0.185	0.473	0.3	6.987	389.115	–	–	0.391	0.08	9.686	10.446	–	–	0.084	0.033
5	12.10267*	1.11597	0.084	0.039	0.064	0.028	0.727	0.236	-0.002	1.531	–	–	0.824	0.251	12.653	27.352	–	–	0.234	0.062
10	8.57382*	0.76679	0.648	0.403	0.588	0.237	0.689	0.208	1.487	1.117	–	–	4.124	1.152	7.647	5.029	–	–	1.027	0.229
15	7.24131*	0.64941	0.499	0.313	0.594	0.301	0.85	0.229	0.33	1.282	–	–	10.031	4.537	7.806	7.356	–	–	1.735	0.332
20	6.30493*	0.56979	1.232	0.793	0.994	0.298	1.075	0.354	0.803	0.492	–	–	14.275	4.417	7.111	3.61	–	–	2.324	0.485
25	5.71335*	0.51724	1.068	0.299	0.989	0.331	1.144	0.532	0.118	0.256	–	–	20.063	10.132	6.67	3.059	–	–	2.98	0.44
Mean:			**0.512**		**0.527**		**0.791**		**1.389**		**–**		**7.136**		**7.749**		**–**		**1.207**	

Table B.16: Clustering details with MiniBooNE Particle Identification (normalized)

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	20	12000	54	1.0	7.5E+06	12000	437	1.0	4.3E+07	12000	594	1.0	4.5E+07	12000	488	0.033	0.967	4.3E+07	4.4E+06	4.1E+06
3	20	12000	56	1.0	1.6E+07	12000	282	1.0	6.7E+07	12000	291	1.0	7.0E+07	12000	260	0.033	0.967	6.9E+07	9.9E+06	7.2E+06
5	20	12000	56	1.0	2.3E+07	12000	194	1.0	9.0E+07	12000	164	1.0	9.1E+07	12000	264	0.167	0.833	8.9E+07	2.3E+07	1.8E+07
10	20	12000	26	1.0	5.5E+07	12000	50	1.0	1.2E+08	12000	72	1.0	1.3E+08	12000	50	0.667	0.333	1.1E+08	1.2E+08	8.6E+07
15	20	12000	14	1.0	7.5E+07	12000	20	1.0	1.3E+08	12000	24	1.0	1.3E+08	12000	23	0.867	0.133	1.2E+08	3.1E+08	1.5E+08
20	20	12000	12	1.0	8.3E+07	12000	14	1.0	1.3E+08	12000	14	1.0	1.3E+08	12000	10	0.233	0.767	1.0E+08	4.2E+08	2.3E+08
25	20	12000	11	1.0	8.8E+07	12000	10	1.0	1.4E+08	12000	12	1.0	1.4E+08	12000	10	0.733	0.267	1.4E+08	6.1E+08	3.0E+08

B.9 MFCCs for Speech Emotion Recognition

Dimensions: $m = 85134$, $n = 58$.

Description: a data set for predicting females and males speech emotions based on Mel Frequency Cepstral Coefficients (MFCCs) values.

Table B.17: Summary of the results with MFCCs for Speech Emotion Recognition ($\times 10^9$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	0.74513*	0.10188	0.029	0.015	0.55	0.262	0.623	0.289	0.04	0.016	0.444	0.279	0.609	0.273	0.04	0.023	0.302	0.222	0.561	0.214
3	0.50215*	0.06923	0.037	0.022	0.352	0.308	0.366	0.297	0.043	0.027	0.186	0.133	0.458	0.3	0.051	0.029	0.25	0.162	0.555	0.184
5	0.3456*	0.04777	0.059	0.03	0.499	0.253	0.499	0.291	0.063	0.043	0.32	0.213	0.579	0.256	0.057	0.022	0.281	0.176	0.596	0.255
10	0.21763*	0.03009	1.209	1.243	0.366	0.133	0.363	0.252	0.11	0.033	0.57	0.22	0.662	0.23	0.129	0.046	0.51	0.165	0.644	0.179
15	0.17608*	0.02458	1.2	0.733	0.301	0.19	0.564	0.205	0.237	0.37	0.49	0.158	0.746	0.21	0.519	0.534	0.464	0.207	0.812	0.218
20	0.15383*	0.0214	0.8	1.017	0.315	0.185	0.706	0.29	0.789	0.397	0.863	0.151	0.903	0.247	0.644	0.364	0.755	0.186	0.982	0.217
25	0.14109*	0.01968	1.142	0.742	0.351	0.218	0.526	0.247	1.09	0.41	1.104	0.165	0.974	0.198	0.97	0.443	0.893	0.368	1.079	0.303
Mean:			**0.639**		**0.39**		**0.521**		**0.339**		**0.568**		**0.704**		**0.344**		**0.494**		**0.747**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	0.74513*	0.10188	0.039	0.02	0.275	0.278	0.345	0.294	0.001	0.0	–	–	0.174	0.065	0.001	0.0	–	–	0.049	0.007
3	0.50215*	0.06923	0.044	0.032	0.242	0.25	0.643	0.261	0.001	0.0	–	–	0.236	0.058	0.001	34.911	–	–	0.069	0.008
5	0.3456*	0.04777	0.057	0.028	0.331	0.22	0.538	0.242	-0.002	0.0	–	–	0.774	0.129	25.789	19.059	–	–	0.2	0.03
10	0.21763*	0.03009	0.102	0.024	0.724	0.222	0.78	0.209	3.278	1.286	–	–	2.347	0.54	11.788	8.119	–	–	0.693	0.145
15	0.17608*	0.02458	0.251	0.263	0.644	0.235	0.855	0.228	1.7	1.843	–	–	5.663	2.193	12.054	8.998	–	–	1.202	0.215
20	0.15383*	0.0214	0.791	0.603	0.832	0.25	0.91	0.344	2.096	1.593	–	–	10.052	2.014	11.16	4.233	–	–	2.035	0.301
25	0.14109*	0.01968	1.017	0.327	0.801	0.166	0.989	0.366	3.385	1.68	–	–	15.119	5.08	9.933	5.571	–	–	2.359	0.407
Mean:			**0.329**		**0.55**		**0.723**		**1.494**		**–**		**4.909**		**10.104**		**–**		**0.944**	

Table B.18: Clustering details with MFCCs for Speech Emotion Recognition

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	20	12000	109	1.0	1.4E+07	12000	469	1.0	6.1E+07	12000	386	1.0	5.7E+07	12000	230	0.767	0.233	5.6E+07	3.5E+06	3.1E+06
3	20	12000	60	1.0	1.9E+07	12000	252	1.0	7.2E+07	12000	312	1.0	7.6E+07	12000	350	0.2	0.8	7.3E+07	5.4E+06	4.8E+06
5	20	12000	54	1.0	2.7E+07	12000	186	1.0	9.1E+07	12000	172	1.0	8.8E+07	12000	150	0.833	0.167	8.9E+07	1.9E+07	1.5E+07
10	20	12000	26	1.0	5.0E+07	12000	67	1.0	1.0E+08	12000	60	1.0	1.1E+08	12000	78	0.967	0.033	1.1E+08	5.7E+07	4.8E+07
15	20	12000	21	1.0	6.0E+07	12000	24	1.0	1.1E+08	12000	30	1.0	1.1E+08	12000	19	0.033	0.967	7.9E+07	1.5E+08	9.9E+07
20	20	12000	20	1.0	6.6E+07	12000	17	1.0	1.2E+08	12000	18	1.0	1.1E+08	12000	13	0.9	0.1	1.1E+08	2.6E+08	1.6E+08
25	20	12000	8	1.0	7.1E+07	12000	10	1.0	1.2E+08	12000	13	1.0	1.2E+08	12000	9	0.833	0.167	1.2E+08	4.2E+08	2.1E+08

B.10 ISOLET

Dimensions: $m = 7797$, $n = 617$.

Description: data set of patterns for spoken letter recognition which contains the spectral coefficients and other additional features.

Table B.19: Summary of the results with ISOLET ($\times 10^5$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7.2194	3.66767	0.033	0.008	1.796	1.551	2.954	1.115	0.033	0.007	1.503	1.185	2.82	1.431	0.032	0.007	0.459	0.692	1.448	1.457
3	6.78782	3.4509	0.054	0.279	0.969	0.694	2.918	1.369	0.044	0.008	2.079	1.157	3.222	0.959	0.043	0.006	0.793	0.573	2.038	1.324
5	6.13651	3.11969	0.456	0.41	0.624	0.341	2.13	1.492	0.066	0.135	1.094	1.663	3.945	1.463	0.071	0.098	0.889	0.608	2.951	1.373
10	5.28577	2.70109	0.622	0.502	0.82	1.018	2.565	1.323	0.189	0.087	1.256	0.335	2.976	1.041	0.343	0.236	0.805	0.342	3.197	1.258
15	4.87391	2.49236	1.4	0.56	0.313	1.402	3.013	1.381	0.647	0.373	1.625	0.748	3.45	1.206	0.552	0.321	1.563	0.437	2.674	0.99
20	4.60857	2.35574	1.162	0.868	1.516	1.047	2.941	1.138	0.357	0.365	2.369	0.887	4.128	0.785	0.391	0.376	2.135	0.599	3.588	0.914
25	4.44323	2.25735	1.0	0.372	1.028	0.139	2.428	1.433	0.28	0.329	3.2	0.591	4.233	1.024	0.332	0.224	3.263	0.714	4.174	0.931
Mean:			**0.675**		**1.01**		**2.707**		**0.231**		**1.875**		**3.539**		**0.252**		**1.415**		**2.867**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7.2194	3.66767	0.032	0.006	1.592	1.149	2.841	1.579	-0.0	0.0	–	–	0.119	0.022	0.026	0.01	–	–	0.062	0.008
3	6.78782	3.4509	0.045	0.007	1.657	1.094	4.006	1.218	0.552	0.27	–	–	0.311	0.283	0.047	0.245	–	–	0.151	0.049
5	6.13651	3.11969	0.07	0.13	1.762	1.292	2.122	1.538	0.392	0.797	–	–	0.63	0.345	0.444	0.691	–	–	0.31	0.087
10	5.28577	2.70109	0.166	0.122	0.949	0.483	3.472	1.262	0.936	1.01	–	–	1.475	1.051	1.281	0.647	–	–	0.609	0.367
15	4.87391	2.49236	0.731	0.354	2.276	1.118	3.789	0.858	1.403	1.382	–	–	1.611	1.074	2.444	1.378	–	–	0.964	0.392
20	4.60857	2.35574	0.34	0.346	2.067	0.664	4.11	0.879	1.079	0.845	–	–	3.058	1.241	1.816	1.161	–	–	1.208	0.389
25	4.44323	2.25735	0.259	0.261	3.407	0.958	4.48	1.001	1.252	0.854	–	–	5.116	0.925	1.127	0.67	–	–	1.989	0.743
Mean:			**0.235**		**1.959**		**3.546**		**0.802**		**–**		**1.76**		**1.026**		**–**		**0.756**	

Table B.20: Clustering details with ISOLET

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	15	4000	283	5.0	1.3E+07	4000	977	5.0	4.9E+07	4000	518	5.0	5.1E+07	4000	1120	1.167	3.833	5.0E+07	1.7E+05	1.1E+05
3	15	4000	240	5.0	1.4E+07	4000	879	5.0	5.4E+07	4000	523	5.0	5.4E+07	4000	1096	1.667	3.333	5.3E+07	4.9E+05	2.9E+05
5	15	4000	128	5.0	1.8E+07	4000	606	5.0	5.6E+07	4000	431	5.0	5.6E+07	4000	300	1.833	3.167	5.5E+07	1.1E+06	4.8E+05
10	15	4000	79	5.0	2.5E+07	4000	147	5.0	5.9E+07	4000	209	5.0	5.9E+07	4000	186	4.667	0.333	5.7E+07	2.5E+06	1.1E+06
15	15	4000	66	5.0	2.8E+07	4000	112	5.0	5.9E+07	4000	58	5.0	5.9E+07	4000	87	1.167	3.833	4.8E+07	2.9E+06	1.9E+06
20	15	4000	40	5.0	3.1E+07	4000	77	5.0	5.8E+07	4000	75	5.0	5.9E+07	4000	80	3.333	1.667	5.7E+07	5.6E+06	2.4E+06
25	15	4000	27	5.0	3.3E+07	4000	43	5.0	5.6E+07	4000	40	5.0	5.8E+07	4000	37	3.167	1.833	4.5E+07	7.8E+06	3.2E+06

B.11 Sensorless Drive Diagnosis

Dimensions: $m = 58509$, $n = 48$.

Description: a data set for sensorless drive diagnosis with features extracted from motor current.

Table B.21: Summary of the results with Sensorless Drive Diagnosis ($\times 10^7$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.88116	3.87915	-0.0	15.678	0.305	0.296	0.392	0.313	-0.0	2.101	0.226	0.151	0.544	0.234	-0.0	1.438	0.26	0.169	0.716	0.257
3	2.91313	3.22719	-0.0	5.899	0.022	0.161	0.516	0.227	-0.0	0.55	0.077	0.008	0.578	0.247	-0.0	0.869	0.082	0.013	0.659	0.267
5	1.93651	1.93613	0.022	8.618	0.307	0.219	0.653	0.285	0.015	7.434	0.559	0.187	0.764	0.219	0.011	1.235	0.48	0.147	0.805	0.184
10	0.98472	1.0394	5.588	8.042	0.177	0.279	0.58	0.257	-2.401	1.407	0.74	0.203	1.017	0.15	-2.394	1.676	0.717	0.179	1.018	0.183
15	0.62816	0.63072	0.481	4.002	0.291	0.251	0.681	0.196	0.034	0.858	1.28	0.448	1.616	0.661	0.028	7.247	1.475	0.412	1.731	0.566
20	0.49884	0.50187	0.486	1.649	0.413	0.135	0.734	0.203	-0.557	1.962	1.78	0.45	2.104	0.596	-0.053	1.871	1.966	0.425	2.326	0.584
25	0.42225	0.43197	2.193	1.768	0.508	0.139	0.811	0.191	1.049	0.546	2.509	0.452	2.867	0.808	0.94	0.502	2.384	0.74	2.826	0.794
Mean:			**1.253**		**0.289**		**0.624**		**-0.266**		**1.024**		**1.356**		**-0.21**		**1.052**		**1.44**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.88116	3.87915	-0.0	0.636	0.213	0.139	0.628	0.276	100.19	0.0	–	–	0.136	0.038	100.19	0.0	–	–	0.14	0.022
3	2.91313	3.22719	-0.0	0.968	0.081	0.014	0.751	0.258	10.865	71.87	–	–	0.433	0.158	10.865	67.653	–	–	0.483	0.152
5	1.93651	1.93613	0.016	6.804	0.704	0.201	0.844	0.207	37.859	0.003	–	–	0.496	0.068	37.853	35.156	–	–	0.524	0.132
10	0.98472	1.0394	-2.404	0.932	0.69	0.194	1.581	0.459	127.202	0.358	–	–	1.696	0.338	127.256	0.036	–	–	1.77	0.387
15	0.62816	0.63072	0.029	0.082	1.339	0.335	1.989	0.587	235.577	0.573	–	–	1.93	0.551	235.435	5.741	–	–	1.789	0.399
20	0.49884	0.50187	-0.058	0.448	1.895	0.392	2.297	0.56	309.27	23.686	–	–	3.769	1.075	309.269	23.374	–	–	3.477	1.446
25	0.42225	0.43197	0.92	0.55	2.407	0.607	2.924	0.944	315.617	35.414	–	–	6.618	2.05	315.673	0.549	–	–	6.369	1.881
Mean:			**-0.214**		**1.047**		**1.573**		**162.369**		–		**2.154**		**162.363**		–		**2.079**	

Table B.22: Clustering details with Sensorless Drive Diagnosis

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	40	58500	32	1.0	1.9E+07	58500	177	1.0	8.5E+07	58500	228	1.0	8.5E+07	58500	199	0.267	0.733	8.4E+07	3.9E+06	3.9E+06
3	40	58500	38	1.0	2.6E+07	58500	134	1.0	9.8E+07	58500	140	1.0	1.0E+08	58500	170	0.2	0.8	9.5E+07	1.4E+07	1.5E+07
5	40	58500	36	1.0	4.0E+07	58500	59	1.0	1.1E+08	58500	84	1.0	1.1E+08	58500	80	0.833	0.167	1.0E+08	1.6E+07	1.7E+07
10	40	58500	18	1.0	5.7E+07	58500	12	1.0	1.3E+08	58500	13	1.0	1.3E+08	58500	10	0.633	0.367	1.2E+08	5.7E+07	5.9E+07
15	40	58500	14	1.0	7.4E+07	58500	5	1.0	2.2E+08	58500	3	1.0	2.3E+08	58500	5	0.4	0.6	2.2E+08	6.6E+07	5.9E+07
20	40	58500	10	1.0	7.7E+07	58500	4	1.0	2.8E+08	58500	5	1.0	2.9E+08	58500	4	0.533	0.467	2.7E+08	1.3E+08	1.2E+08
25	40	58500	7	1.0	8.3E+07	58500	5	1.0	3.7E+08	58500	4	1.0	3.5E+08	58500	5	0.767	0.233	3.6E+08	2.2E+08	2.1E+08

B.12 Sensorless Drive Diagnosis (normalized)

Dimensions: $m = 58509$, $n = 48$.

Description: a data set for sensorless drive diagnosis with features extracted from motor current. Min-max scaling was used for normalization of data set values for better clusterization.

Table B.23: Summary of the results with Sensorless Drive Diagnosis (normalized) ($\times 10^3$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	15.64798*	0.89303	0.067	0.035	0.1	0.082	0.122	0.08	0.091	0.046	0.073	0.052	0.122	0.082	0.088	0.051	0.076	0.069	0.15	0.087
3	12.19375*	0.70587	3.467	2.324	0.058	0.097	0.139	0.098	0.187	1.016	0.054	0.053	0.159	0.076	0.187	1.145	0.044	0.044	0.163	0.091
5	7.85054*	0.45202	0.363	1.748	0.099	0.078	0.166	0.076	0.343	0.255	0.066	0.058	0.172	0.08	0.293	0.21	0.056	0.062	0.181	0.087
10	4.71275*	0.28067	3.764	2.034	0.089	0.08	0.165	0.08	0.609	1.073	0.067	0.058	0.212	0.074	1.936	1.295	0.064	0.038	0.201	0.075
15	3.62541*	0.21493	3.765	2.962	0.106	0.07	0.229	0.091	1.445	0.992	0.111	0.048	0.203	0.072	1.85	1.395	0.091	0.045	0.223	0.071
20	2.971*	0.17797	4.762	2.238	0.059	0.068	0.169	0.087	2.142	0.786	0.101	0.051	0.23	0.065	2.391	1.266	0.099	0.034	0.233	0.066
25	2.60929*	0.15364	5.017	2.274	0.111	0.065	0.2	0.086	2.629	1.204	0.155	0.078	0.25	0.076	2.993	1.446	0.185	0.07	0.246	0.061
Mean:			**3.029**		**0.089**		**0.17**		**1.064**		**0.09**		**0.193**		**1.391**		**0.088**		**0.2**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	15.64798*	0.89303	0.089	0.059	0.067	0.081	0.175	0.086	0.0	11.811	–	–	0.051	0.019	0.002	17.132	–	–	0.012	0.001
3	12.19375*	0.70587	0.15	0.596	0.078	0.065	0.191	0.074	0.979	3.288	–	–	0.09	0.066	1.574	11.101	–	–	0.033	0.008
5	7.85054*	0.45202	0.297	0.238	0.06	0.044	0.157	0.087	0.535	2.44	–	–	0.21	0.15	11.592	16.743	–	–	0.055	0.012
10	4.71275*	0.28067	1.148	1.18	0.062	0.051	0.253	0.08	6.68	3.559	–	–	0.563	0.402	13.32	7.63	–	–	0.131	0.025
15	3.62541*	0.21493	1.781	0.887	0.094	0.056	0.244	0.079	8.774	3.827	–	–	0.978	0.407	14.032	8.194	–	–	0.201	0.032
20	2.971*	0.17797	3.125	1.012	0.101	0.063	0.251	0.073	12.594	5.298	–	–	1.644	0.589	18.915	6.425	–	–	0.265	0.051
25	2.60929*	0.15364	2.768	1.106	0.161	0.072	0.275	0.065	13.879	6.179	–	–	1.806	0.741	19.277	4.994	–	–	0.318	0.05
Mean:			**1.337**		**0.089**		**0.221**		**6.206**		**–**		**0.763**		**11.245**		**–**		**0.145**	

Table B.24: Clustering details with Sensorless Drive Diagnosis (normalized)

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	40	3500	38	0.3	2.2E+06	3500	252	0.3	1.4E+07	3500	298	0.3	1.4E+07	3500	362	0.16	0.14	1.4E+07	1.2E+06	1.1E+06
3	40	3500	42	0.3	3.4E+06	3500	248	0.3	1.9E+07	3500	293	0.3	2.0E+07	3500	321	0.13	0.17	1.9E+07	2.7E+06	2.6E+06
5	40	3500	36	0.3	5.4E+06	3500	174	0.3	2.6E+07	3500	194	0.3	2.6E+07	3500	158	0.01	0.29	2.5E+07	5.9E+06	5.4E+06
10	40	3500	27	0.3	9.7E+06	3500	110	0.3	3.4E+07	3500	114	0.3	3.2E+07	3500	134	0.23	0.07	3.3E+07	1.9E+07	1.4E+07
15	40	3500	31	0.3	1.5E+07	3500	52	0.3	3.7E+07	3500	60	0.3	3.7E+07	3500	70	0.26	0.04	3.6E+07	3.2E+07	2.3E+07
20	40	3500	20	0.3	1.8E+07	3500	39	0.3	3.9E+07	3500	40	0.3	3.8E+07	3500	34	0.1	0.2	3.4E+07	5.6E+07	2.9E+07
25	40	3500	20	0.3	2.1E+07	3500	30	0.3	4.0E+07	3500	28	0.3	3.9E+07	3500	28	0.1	0.2	3.6E+07	6.2E+07	3.7E+07

B.13 Online News Popularity

Dimensions: $m = 39644$, $n = 58$.

Description: this dataset summarizes a heterogeneous set of features about articles published by Mashable in a period of two years for predicting the number of shares in social networks (popularity).

Table B.25: Summary of the results with Online News Popularity ($\times 10^{14}$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	9.53913	2.23789	0.012	0.011	0.328	0.146	0.345	0.177	0.023	0.015	0.134	0.131	0.254	0.164	0.019	0.009	0.064	0.098	0.187	0.132
3	5.91077	1.35797	0.05	7.212	0.403	0.156	0.357	0.174	0.061	0.035	0.211	0.175	0.392	0.219	0.072	0.031	0.256	0.146	0.415	0.164
5	3.09885	0.70224	0.08	6.232	0.473	0.147	0.483	0.162	0.068	0.018	0.193	0.101	0.34	0.186	0.08	0.034	0.146	0.093	0.374	0.181
10	1.17247	0.27667	3.005	5.598	0.198	0.172	0.359	0.188	1.531	0.834	0.265	0.131	0.571	0.178	1.001	1.526	0.267	0.128	0.424	0.126
15	0.77637	0.1913	2.99	5.129	0.149	0.162	0.468	0.167	2.225	1.195	0.265	0.156	0.473	0.171	1.863	1.21	0.259	0.123	0.564	0.161
20	0.59809	0.14447	4.752	2.196	0.156	0.147	0.441	0.226	2.587	1.268	0.406	0.122	0.568	0.152	3.388	1.168	0.418	0.15	0.552	0.167
25	0.49616	0.1202	5.599	1.786	0.205	0.161	0.262	0.24	5.083	2.276	0.551	0.147	0.62	0.149	4.767	7.225	0.529	0.149	0.594	0.16
Mean:			**2.355**		**0.273**		**0.388**		**1.654**		**0.289**		**0.46**		**1.598**		**0.277**		**0.444**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	9.53913	2.23789	0.015	0.008	0.131	0.128	0.316	0.158	-0.0	0.0	–	–	0.036	0.015	0.001	0.0	–	–	0.017	0.004
3	5.91077	1.35797	0.089	0.033	0.21	0.182	0.495	0.193	0.0	0.0	–	–	0.206	0.062	1.679	29.518	–	–	0.057	0.015
5	3.09885	0.70224	0.076	0.026	0.187	0.136	0.378	0.147	12.069	7.238	–	–	0.253	0.167	80.754	51.871	–	–	0.117	0.046
10	1.17247	0.27667	0.928	0.717	0.276	0.177	0.581	0.186	12.363	17.442	–	–	0.877	0.705	38.376	20.585	–	–	0.511	0.163
15	0.77637	0.1913	1.91	3.185	0.355	0.2	0.611	0.24	16.822	9.168	–	–	1.898	0.58	43.769	12.082	–	–	0.819	0.229
20	0.59809	0.14447	3.687	1.27	0.395	0.204	0.675	0.297	25.504	6.434	–	–	3.253	1.371	46.453	19.54	–	–	1.822	0.357
25	0.49616	0.1202	5.135	5.994	0.495	0.14	0.643	0.248	37.787	10.213	–	–	7.378	2.996	53.319	12.025	–	–	2.015	0.471
Mean:			**1.692**		**0.293**		**0.529**		**14.935**		–		**1.986**		**37.764**		–		**0.765**	

Table B.26: Clustering details with Online News Popularity

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	20	10000	98	0.7	8.7E+06	10000	382	0.7	4.5E+07	10000	281	0.7	4.4E+07	10000	455	0.63	0.07	4.5E+07	5.6E+05	4.4E+05
3	20	10000	54	0.7	1.3E+07	10000	266	0.7	5.9E+07	10000	258	0.7	6.2E+07	10000	312	0.14	0.56	6.0E+07	4.9E+06	2.0E+06
5	20	10000	74	0.7	2.0E+07	10000	130	0.7	6.6E+07	10000	191	0.7	6.8E+07	10000	168	0.467	0.233	6.6E+07	6.0E+06	4.1E+06
10	20	10000	32	0.7	3.0E+07	10000	97	0.7	7.3E+07	10000	60	0.7	7.5E+07	10000	84	0.49	0.21	6.6E+07	2.4E+07	1.9E+07
15	20	10000	26	0.7	4.2E+07	10000	26	0.7	7.7E+07	10000	33	0.7	7.9E+07	10000	14	0.047	0.653	5.7E+07	4.8E+07	3.4E+07
20	20	10000	14	0.7	4.7E+07	10000	16	0.7	8.2E+07	10000	16	0.7	8.0E+07	10000	20	0.63	0.07	8.1E+07	9.1E+07	6.6E+07
25	20	10000	5	0.7	5.1E+07	10000	10	0.7	8.4E+07	10000	11	0.7	8.5E+07	10000	13	0.653	0.047	8.2E+07	2.1E+08	8.2E+07

B.14 Gas Sensor Array Drift

Dimensions: $m = 13910$, $n = 128$.

Description: this data set contains measurements from chemical sensors utilized in simulations for drift compensation in a discrimination task of different gases at various levels of concentrations.

Table B.27: Summary of the results with Gas Sensor Array Drift ($\times 10^{13}$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7.91186	4.78811	0.088	0.064	0.46	0.394	0.844	0.552	0.127	0.091	0.438	0.614	1.283	0.614	0.203	0.084	0.462	0.576	1.118	0.573
3	5.02412	3.01636	0.133	9.866	0.627	0.548	0.896	0.572	0.194	0.11	0.44	0.398	0.924	0.582	0.255	0.109	0.631	0.445	1.119	0.582
5	3.22394	2.03175	6.935	3.508	0.727	0.638	0.797	0.572	7.107	3.625	0.718	0.48	1.151	0.544	0.178	1.763	0.589	0.446	1.145	0.525
10	1.65524	1.06767	3.155	2.805	0.169	0.367	0.797	0.527	0.434	1.262	0.325	0.167	1.193	0.46	0.274	1.403	0.285	0.214	1.128	0.562
15	1.13801	0.74507	4.665	3.27	0.192	0.362	1.024	0.552	0.202	1.306	0.453	0.199	1.269	0.492	0.418	1.634	0.43	0.095	1.307	0.445
20	0.87916	0.56988	3.129	2.638	0.717	0.544	1.319	0.604	1.84	0.792	1.023	0.415	1.488	0.422	2.375	1.02	0.853	0.342	1.488	0.396
25	0.72274	0.47044	4.598	1.838	0.396	0.568	1.008	0.548	2.526	0.797	1.42	0.45	1.685	0.492	2.684	0.897	1.165	0.357	1.528	0.406
Mean:			**3.243**		**0.47**		**0.955**		**1.776**		**0.688**		**1.285**		**0.912**		**0.631**		**1.262**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7.91186	4.78811	0.192	0.1	0.394	0.516	0.952	0.654	-0.0	0.0	–	–	0.053	0.007	0.029	0.048	–	–	0.028	0.009
3	5.02412	3.01636	0.245	0.11	0.451	0.523	1.044	0.566	-0.001	0.0	–	–	0.104	0.016	0.03	0.034	–	–	0.088	0.024
5	3.22394	2.03175	0.366	3.35	0.817	0.525	1.245	0.557	8.108	0.387	–	–	0.247	0.064	8.156	0.394	–	–	0.118	0.03
10	1.65524	1.06767	0.232	0.696	0.287	0.21	1.592	0.468	37.905	17.254	–	–	0.595	0.415	41.32	13.482	–	–	0.371	0.23
15	1.13801	0.74507	-0.175	1.603	0.46	0.184	1.887	0.59	27.472	10.362	–	–	1.653	0.679	30.525	24.91	–	–	0.704	0.319
20	0.87916	0.56988	2.03	1.033	1.132	0.391	1.673	0.48	45.732	7.967	–	–	1.684	0.509	45.904	6.855	–	–	0.982	0.472
25	0.72274	0.47044	2.639	0.954	1.54	0.425	1.951	0.446	50.936	12.149	–	–	2.544	1.197	52.691	14.414	–	–	1.796	0.664
Mean:			**0.79**		**0.726**		**1.478**		**24.307**		–		**0.983**		**25.522**		–		**0.584**	

Table B.28: Clustering details with Gas Sensor Array Drift

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	30	9000	140	2.0	2.3E+07	9000	810	2.0	9.0E+07	9000	606	2.0	9.2E+07	9000	642	1.0	1.0	9.2E+07	5.0E+05	2.8E+05
3	30	9000	128	2.0	2.9E+07	9000	424	2.0	9.6E+07	9000	468	2.0	9.9E+07	9000	544	0.867	1.133	1.0E+08	1.0E+06	6.2E+05
5	30	9000	84	2.0	3.8E+07	9000	300	2.0	1.0E+08	9000	292	2.0	1.0E+08	9000	292	0.867	1.133	9.8E+07	2.5E+06	1.3E+06
10	30	9000	46	2.0	5.5E+07	9000	102	2.0	1.1E+08	9000	106	2.0	1.1E+08	9000	144	1.4	0.6	1.0E+08	6.1E+06	3.9E+06
15	30	9000	42	2.0	5.8E+07	9000	65	2.0	1.1E+08	9000	65	2.0	1.1E+08	9000	96	1.867	0.133	1.0E+08	1.6E+07	7.6E+06
20	30	9000	40	2.0	6.2E+07	9000	44	2.0	1.1E+08	9000	38	2.0	1.1E+08	9000	30	0.2	1.8	7.9E+07	1.8E+07	1.1E+07
25	30	9000	20	2.0	6.5E+07	9000	29	2.0	1.1E+08	9000	30	2.0	1.1E+08	9000	16	0.8	1.2	7.3E+07	2.7E+07	1.9E+07

B.15 3D Road Network

Dimensions: $m = 434874$, $n = 3$.

Description: 3D road network from Denmark with highly accurate elevation information which contains longitude, latitude and altitude for each road segment or edge in the graph. Usually this data set used in eco-routing and fuel/Co2-estimation routing algorithms.

Table B.29: Summary of the results with 3D Road Network ($\times 10^6$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	49.13298	11.15303	0.004	0.006	0.257	0.123	0.265	0.125	0.005	0.015	0.195	0.151	0.304	0.149	0.008	0.005	0.177	0.107	0.228	0.131
3	22.77818	5.1707	0.005	0.007	0.19	0.133	0.318	0.141	0.011	0.011	0.15	0.102	0.285	0.145	0.015	0.015	0.13	0.093	0.263	0.145
5	8.82574	1.99891	0.02	0.014	0.182	0.146	0.294	0.136	0.021	0.021	0.249	0.115	0.341	0.121	0.018	0.024	0.182	0.099	0.346	0.139
10	2.56661	0.58256	0.167	0.116	0.167	0.113	0.234	0.122	0.159	0.103	0.301	0.137	0.418	0.142	0.164	0.185	0.315	0.099	0.419	0.121
15	1.27069	0.28889	0.334	0.38	0.276	0.104	0.382	0.144	0.223	0.377	0.502	0.346	0.504	0.332	0.343	0.331	0.442	0.198	0.503	0.193
20	0.80865	0.18573	1.243	0.823	0.287	0.109	0.343	0.098	0.542	0.652	0.46	0.168	0.591	0.184	0.382	0.644	0.463	0.208	0.541	0.191
25	0.59259	0.13625	1.038	0.84	0.242	0.13	0.405	0.118	0.557	0.489	0.603	0.264	0.757	0.351	0.588	0.487	0.603	0.255	0.755	0.276
Mean:			**0.402**		**0.229**		**0.32**		**0.217**		**0.351**		**0.457**		**0.217**		**0.33**		**0.436**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	49.13298	11.15303	0.01	0.011	0.157	0.115	0.195	0.154	0.0	0.0	–	–	0.113	0.033	0.0	0.0	–	–	0.062	0.007
3	22.77818	5.1707	0.012	0.012	0.123	0.082	0.233	0.121	0.0	0.0	–	–	0.187	0.065	0.0	77.393	–	–	0.106	0.009
5	8.82574	1.99891	0.031	0.027	0.215	0.106	0.309	0.129	0.0	0.0	–	–	0.517	0.087	77.246	43.749	–	–	0.268	0.034
10	2.56661	0.58256	0.227	0.176	0.546	0.188	0.503	0.196	0.008	0.0	–	–	5.994	0.261	62.553	44.418	–	–	1.802	0.34
15	1.27069	0.28889	0.224	0.25	0.455	0.443	0.501	0.48	0.002	0.0	–	–	7.558	0.788	57.087	42.217	–	–	2.768	0.357
20	0.80865	0.18573	0.468	0.501	0.43	0.315	0.797	0.418	0.005	0.0	–	–	25.175	1.893	42.58	23.013	–	–	4.995	0.876
25	0.59259	0.13625	0.523	0.481	0.685	0.483	0.874	0.6	1.615	0.25	–	–	24.839	1.744	45.092	33.97	–	–	6.186	0.732
Mean:			**0.214**		**0.373**		**0.487**		**0.233**		–		**9.198**		**40.651**		–		**2.313**	

Table B.30: Clustering details with 3D Road Network

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	40	100000	17	0.5	1.8E+07	100000	100	0.5	9.6E+07	100000	72	0.5	9.6E+07	100000	57	0.033	0.467	1.0E+08	2.1E+07	1.8E+07
3	40	100000	15	0.5	2.6E+07	100000	84	0.5	1.6E+08	100000	78	0.5	1.6E+08	100000	68	0.467	0.033	1.6E+08	4.2E+07	4.2E+07
5	40	100000	14	0.5	5.7E+07	100000	74	0.5	2.8E+08	100000	71	0.5	2.7E+08	100000	52	0.117	0.383	2.7E+08	1.5E+08	1.3E+08
10	40	100000	8	0.5	1.7E+08	100000	26	0.5	6.0E+08	100000	24	0.5	5.6E+08	100000	10	0.15	0.35	4.3E+08	2.2E+09	1.2E+09
15	40	100000	6	0.5	2.6E+08	100000	8	0.5	8.5E+08	100000	10	0.5	8.1E+08	100000	11	0.45	0.05	7.9E+08	3.0E+09	2.4E+09
20	40	100000	6	0.5	3.6E+08	100000	6	0.5	9.5E+08	100000	6	0.5	9.9E+08	100000	6	0.333	0.167	9.7E+08	1.1E+10	5.0E+09
25	40	100000	5	0.5	3.6E+08	100000	5	0.5	1.2E+09	100000	5	0.5	1.3E+09	100000	5	0.317	0.183	1.2E+09	1.1E+10	7.8E+09

B.16 Skin Segmentation

Dimensions: $m = 245057$, $n = 3$.

Description: Skin and Nonskin dataset is generated using skin textures from face images of diversity of age, gender, and race people and constructed over B, G, R color space.

Table B.31: Summary of the results with Skin Segmentation ($\times 10^9$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	1.32236	0.04216	0.031	0.019	0.105	0.045	0.097	0.054	0.035	0.013	0.04	0.026	0.105	0.055	0.034	0.022	0.028	0.041	0.106	0.05
3	0.89362	0.02822	0.054	0.032	0.058	0.052	0.084	0.06	0.043	0.024	0.066	0.035	0.099	0.049	0.038	0.03	0.038	0.046	0.107	0.065
5	0.50203	0.0161	0.124	2.491	0.048	0.046	0.134	0.053	0.073	0.586	0.018	0.013	0.143	0.056	0.078	0.815	0.018	0.025	0.104	0.051
10	0.25121	0.00817	6.804	5.439	0.039	0.075	0.113	0.061	0.212	1.399	0.026	0.016	0.142	0.053	0.247	2.335	0.023	0.037	0.136	0.064
15	0.16964	0.00544	3.665	2.287	0.064	0.044	0.128	0.059	1.201	1.962	0.046	0.029	0.142	0.042	0.734	2.771	0.038	0.032	0.12	0.053
20	0.12615	0.004	4.366	2.928	0.11	0.055	0.126	0.054	2.311	1.567	0.092	0.051	0.138	0.051	2.202	2.534	0.07	0.034	0.121	0.047
25	0.10228	0.00335	5.333	2.735	0.067	0.035	0.104	0.051	3.485	1.754	0.052	0.016	0.15	0.051	4.461	1.755	0.056	0.034	0.155	0.046
Mean:			**2.911**		**0.07**		**0.112**		**1.052**		**0.049**		**0.131**		**1.113**		**0.039**		**0.121**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	1.32236	0.04216	0.036	0.022	0.034	0.04	0.144	0.057	-0.0	0.0	–	–	0.042	0.008	-0.0	0.007	–	–	0.014	0.001
3	0.89362	0.02822	0.069	0.031	0.041	0.042	0.106	0.054	-0.001	0.0	–	–	0.081	0.032	0.003	59.847	–	–	0.025	0.003
5	0.50203	0.0161	0.092	0.298	0.021	0.022	0.124	0.062	1.65	6.344	–	–	0.117	0.036	18.075	22.33	–	–	0.036	0.003
10	0.25121	0.00817	0.202	2.214	0.029	0.03	0.176	0.059	9.122	7.003	–	–	0.219	0.062	26.085	8.432	–	–	0.062	0.007
15	0.16964	0.00544	0.888	2.013	0.04	0.039	0.152	0.051	13.463	7.936	–	–	0.389	0.187	29.36	13.524	–	–	0.104	0.011
20	0.12615	0.004	2.121	1.667	0.089	0.04	0.151	0.033	16.816	7.379	–	–	0.548	0.224	34.997	18.226	–	–	0.145	0.018
25	0.10228	0.00335	3.727	1.445	0.061	0.023	0.164	0.049	22.066	6.921	–	–	0.698	0.237	35.345	16.71	–	–	0.182	0.023
Mean:			**1.019**		**0.045**		**0.145**		**9.016**		**–**		**0.299**		**20.552**		**–**		**0.081**	

Table B.32: Clustering details with Skin Segmentation

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	30	8000	10	0.2	1.6E+06	8000	89	0.2	9.8E+06	8000	80	0.2	8.8E+06	8000	114	0.047	0.153	9.4E+06	6.9E+06	5.4E+06
3	30	8000	10	0.2	3.5E+06	8000	75	0.2	1.8E+07	8000	82	0.2	1.7E+07	8000	76	0.033	0.167	1.8E+07	1.8E+07	1.7E+07
5	30	8000	16	0.2	5.3E+06	8000	111	0.2	2.8E+07	8000	78	0.2	2.5E+07	8000	88	0.153	0.047	2.7E+07	2.7E+07	2.4E+07
10	30	8000	8	0.2	1.1E+07	8000	81	0.2	6.3E+07	8000	90	0.2	5.9E+07	8000	96	0.127	0.073	6.0E+07	7.6E+07	6.5E+07
15	30	8000	14	0.2	2.4E+07	8000	64	0.2	1.1E+08	8000	53	0.2	1.0E+08	8000	72	0.153	0.047	1.1E+08	1.5E+08	1.3E+08
20	30	8000	11	0.2	3.0E+07	8000	48	0.2	1.5E+08	8000	44	0.2	1.3E+08	8000	56	0.053	0.147	1.3E+08	2.3E+08	1.8E+08
25	30	8000	10	0.2	4.6E+07	8000	44	0.2	1.5E+08	8000	48	0.2	1.5E+08	8000	48	0.16	0.04	1.5E+08	2.8E+08	2.4E+08

B.17 KEGG Metabolic Relation Network (Directed)

Dimensions: $m = 53413$, $n = 20$.

Description:

Table B.33: Summary of the results with KEGG Metabolic Relation Network (Directed) ($\times 10^8$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	11.3853	11.29955	0.0	8.626	0.246	0.336	0.434	0.282	0.0	0.115	0.277	0.242	0.655	0.273	0.24	0.115	0.112	0.116	0.384	0.289
3	4.9006	4.84007	0.001	27.183	0.296	0.183	0.486	0.226	0.559	0.242	0.201	0.179	0.4	0.252	0.559	0.277	0.226	0.235	0.69	0.272
5	1.88367	1.86304	0.005	0.315	0.521	0.276	0.585	0.292	0.016	0.708	0.321	0.196	0.499	0.238	0.014	0.707	0.381	0.198	0.557	0.232
10	0.60513	0.61753	0.07	7.977	0.077	0.226	0.681	0.307	0.022	1.556	0.269	0.083	0.683	0.174	0.041	0.024	0.254	0.17	0.643	0.244
15	0.35393	0.35466	4.554	6.115	0.591	0.25	0.538	0.182	-0.418	0.998	0.451	0.196	0.87	0.223	-0.491	2.633	0.387	0.164	0.853	0.198
20	0.25027	0.25131	2.103	6.812	0.152	0.267	0.76	0.3	0.149	0.63	0.799	0.198	1.006	0.195	0.433	0.795	0.792	0.213	0.966	0.272
25	0.19289	0.19795	4.091	2.5	0.217	0.155	0.545	0.217	1.372	1.097	0.914	0.313	1.143	0.284	1.64	5.875	0.818	0.233	1.064	0.236
Mean:			**1.546**		**0.3**		**0.575**		**0.243**		**0.462**		**0.751**		**0.348**		**0.424**		**0.737**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	11.3853	11.29955	0.0	0.11	0.249	0.126	0.477	0.233	18.854	0.0	–	–	0.041	0.004	18.854	0.003	–	–	0.033	0.006
3	4.9006	4.84007	0.559	0.256	0.226	0.218	0.373	0.297	124.789	0.0	–	–	0.08	0.023	124.789	9.606	–	–	0.075	0.008
5	1.88367	1.86304	0.072	0.715	0.376	0.249	0.731	0.205	0.0	9.787	–	–	0.201	0.042	0.0	0.001	–	–	0.189	0.07
10	0.60513	0.61753	0.041	8.926	0.276	0.056	0.766	0.194	36.81	3.376	–	–	0.607	0.158	36.81	3.067	–	–	0.582	0.039
15	0.35393	0.35466	-0.359	1.22	0.504	0.186	0.935	0.211	96.641	4.224	–	–	1.873	0.168	97.957	4.103	–	–	1.69	0.245
20	0.25027	0.25131	0.15	25.29	0.631	0.364	1.26	0.303	162.301	4.756	–	–	3.433	0.856	162.039	3.883	–	–	3.784	0.849
25	0.19289	0.19795	1.312	0.908	0.814	0.224	1.421	0.32	230.281	6.699	–	–	5.045	0.638	223.96	5.88	–	–	5.15	0.602
Mean:			**0.254**		**0.44**		**0.852**		**95.668**		–		**1.611**		**94.916**		–		**1.643**	

Table B.34: Clustering details with KEGG Metabolic Relation Network (Directed)

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	20	53350	54	1.0	3.1E+07	53350	586	1.0	2.0E+08	53350	348	1.0	2.0E+08	53350	377	0.767	0.233	1.9E+08	1.9E+06	2.0E+06
3	20	53350	62	1.0	4.4E+07	53350	222	1.0	2.3E+08	53350	412	1.0	2.2E+08	53350	206	0.967	0.033	2.3E+08	5.4E+06	5.4E+06
5	20	53350	64	1.0	7.4E+07	53350	144	1.0	2.6E+08	53350	190	1.0	2.8E+08	53350	260	0.333	0.667	2.7E+08	1.5E+07	1.5E+07
10	20	53350	52	1.0	1.1E+08	53350	77	1.0	2.8E+08	53350	73	1.0	3.0E+08	53350	108	0.867	0.133	2.7E+08	5.4E+07	5.3E+07
15	20	53350	34	1.0	1.3E+08	53350	62	1.0	2.9E+08	53350	56	1.0	3.0E+08	53350	35	0.6	0.4	2.3E+08	1.7E+08	1.5E+08
20	20	53350	34	1.0	1.6E+08	53350	31	1.0	3.0E+08	53350	27	1.0	2.9E+08	53350	13	0.1	0.9	2.3E+08	3.2E+08	3.5E+08
25	20	53350	18	1.0	1.7E+08	53350	14	1.0	3.1E+08	53350	12	1.0	3.1E+08	53350	10	0.033	0.967	2.8E+08	4.6E+08	4.8E+08

B.18 Shuttle Control

Dimensions: $m = 58000$, $n = 9$.

Description: each entity in the dataset contains several shuttle control attributes.

Table B.35: Summary of the results with Shuttle Control ($\times 10^8$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	21.34329	19.86153	5.043	12.056	0.724	0.333	0.716	0.364	5.043	1.082	0.22	0.357	1.0	0.34	0.0	0.744	0.093	0.067	0.821	0.37
3	10.85415	10.49161	0.28	29.163	0.532	0.425	0.852	0.398	3.658	1.293	0.418	0.338	1.02	0.48	3.658	1.613	0.418	0.308	0.854	0.311
4	8.8691	8.62423	0.32	4.741	0.816	0.342	0.695	0.397	0.343	7.623	0.793	0.417	0.879	0.447	0.0	0.075	0.307	0.361	0.687	0.446
5	7.24479	7.28912	1.484	7.359	0.017	0.037	0.73	0.456	0.178	7.394	0.034	0.077	0.757	0.403	0.392	0.21	0.033	0.005	0.714	0.429
10	2.83216	2.99551	8.736	21.835	0.148	0.337	0.859	0.422	1.623	2.889	0.082	0.011	0.944	0.438	0.671	0.475	0.081	0.012	0.412	0.399
15	1.53154	1.69671	16.164	8.425	0.053	0.289	0.883	0.411	5.617	2.582	0.149	0.022	0.738	0.363	5.814	2.605	0.146	0.016	1.054	0.427
20	1.06012	1.07621	3.493	7.041	0.181	0.409	0.952	0.41	-0.758	3.626	0.225	0.045	1.102	0.419	-1.494	2.123	0.21	0.079	1.04	0.437
25	0.77978	0.79776	9.944	4.377	0.083	0.0	0.688	0.394	2.84	3.246	0.378	0.361	1.212	0.35	3.339	3.844	0.387	0.311	1.173	0.268
Mean:			**5.683**		**0.319**		**0.797**		**2.318**		**0.287**		**0.957**		**1.548**		**0.209**		**0.844**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	21.34329	19.86153	1.86	2.186	0.115	0.269	0.916	0.412	51.112	11.489	–	–	0.022	0.007	51.112	0.025	–	–	0.026	0.006
3	10.85415	10.49161	3.658	1.368	0.245	0.183	0.553	0.373	100.557	38.781	–	–	0.07	0.053	100.558	44.75	–	–	0.041	0.039
4	8.8691	8.62423	0.0	7.064	0.204	0.374	0.36	0.463	143.415	59.951	–	–	0.057	0.032	143.415	50.661	–	–	0.043	0.019
5	7.24479	7.28912	0.178	5.719	0.032	0.003	0.767	0.352	38.691	61.282	–	–	0.058	0.018	38.774	47.006	–	–	0.074	0.022
10	2.83216	2.99551	0.692	0.98	0.084	0.008	0.884	0.3	135.103	37.214	–	–	0.137	0.079	135.73	31.725	–	–	0.177	0.07
15	1.53154	1.69671	3.768	2.945	0.145	0.016	1.193	0.401	225.668	38.69	–	–	0.228	0.102	243.615	42.423	–	–	0.213	0.051
20	1.06012	1.07621	0.017	2.21	0.226	0.256	1.054	0.4	324.175	40.507	–	–	0.308	0.078	284.79	24.232	–	–	0.275	0.09
25	0.77978	0.79776	4.719	2.264	0.654	0.328	1.21	0.254	391.295	22.132	–	–	0.674	0.23	396.372	19.782	–	–	0.555	0.232
Mean:			**1.862**		**0.213**		**0.867**		**176.252**		**–**		**0.194**		**174.296**		**–**		**0.176**	

Table B.36: Clustering details with Shuttle Control

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	15	57950	200	1.5	9.1E+07	57950	1378	1.5	5.2E+08	57950	1169	1.5	5.3E+08	57950	1370	0.75	0.75	5.3E+08	2.8E+06	3.2E+06
3	15	57950	175	1.5	1.1E+08	57950	953	1.5	6.0E+08	57950	1008	1.5	6.3E+08	57950	618	0.65	0.85	6.2E+08	6.1E+06	5.9E+06
4	15	57950	139	1.5	1.4E+08	57950	835	1.5	7.2E+08	57950	681	1.5	6.8E+08	57950	338	0.6	0.9	7.1E+08	7.9E+06	7.2E+06
5	15	57950	145	1.5	1.8E+08	57950	600	1.5	7.6E+08	57950	568	1.5	7.8E+08	57950	643	0.6	0.9	8.0E+08	9.3E+06	8.1E+06
10	15	57950	106	1.5	2.8E+08	57950	432	1.5	9.6E+08	57950	172	1.5	9.2E+08	57950	416	0.1	1.4	9.0E+08	2.8E+07	3.6E+07
15	15	57950	105	1.5	3.8E+08	57950	204	1.5	9.4E+08	57950	310	1.5	1.0E+09	57950	336	1.1	0.4	9.6E+08	4.9E+07	4.5E+07
20	15	57950	80	1.5	4.5E+08	57950	186	1.5	9.8E+08	57950	157	1.5	9.7E+08	57950	146	0.6	0.9	9.7E+08	6.4E+07	5.9E+07
25	15	57950	56	1.5	5.0E+08	57950	157	1.5	9.8E+08	57950	135	1.5	1.0E+09	57950	118	0.35	1.15	9.1E+08	1.0E+08	1.2E+08

B.19 Shuttle Control (normalized)

Dimensions: $m = 58000$, $n = 9$.

Description: each entity in the dataset contains several shuttle control attributes. Min-max scaling was used for normalization of data set values for better clusterization.

Table B.37: Summary of the results with Shuttle Control (normalized) ($\times 10^1$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	104.41601	3.33677	0.106	0.211	0.244	0.103	0.218	0.101	0.184	0.167	0.068	0.065	0.164	0.103	0.245	0.102	0.112	0.078	0.182	0.122
3	73.28769	2.33445	0.697	0.868	0.132	0.065	0.135	0.129	0.675	0.297	0.029	0.049	0.18	0.111	0.514	0.519	0.039	0.036	0.153	0.1
4	50.076	1.5748	0.781	12.197	0.212	0.116	0.242	0.116	0.675	0.508	0.019	0.03	0.204	0.111	0.46	0.347	0.023	0.026	0.132	0.109
5	39.78043	1.24889	1.301	1.451	0.057	0.068	0.189	0.134	1.224	0.875	0.023	0.02	0.248	0.121	1.679	0.823	0.019	0.013	0.149	0.116
10	15.04997	0.44476	2.315	11.582	0.215	0.147	0.245	0.123	0.824	0.969	0.143	0.1	0.297	0.088	2.23	0.96	0.057	0.114	0.267	0.095
15	9.81804	0.28928	5.001	3.919	0.066	0.094	0.25	0.111	3.02	1.72	0.042	0.018	0.217	0.096	2.906	1.421	0.027	0.025	0.26	0.116
20	7.233	0.19874	6.611	3.444	0.114	0.099	0.243	0.102	2.84	1.499	0.062	0.049	0.244	0.102	4.49	2.5	0.051	0.043	0.245	0.118
25	5.86461	0.15054	5.645	3.749	0.14	0.13	0.207	0.126	4.909	1.212	0.14	0.076	0.255	0.108	5.227	1.257	0.094	0.088	0.246	0.101
Mean:			**2.807**		**0.147**		**0.216**		**1.794**		**0.066**		**0.226**		**2.219**		**0.053**		**0.204**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	104.41601	3.33677	0.231	0.153	0.148	0.084	0.221	0.124	14.732	10.172	–	–	0.018	0.005	7.998	2.641	–	–	0.006	0.001
3	73.28769	2.33445	0.505	0.279	0.07	0.091	0.305	0.107	1.765	12.847	–	–	0.027	0.007	14.401	16.099	–	–	0.008	0.001
4	50.076	1.5748	0.514	0.415	0.025	0.058	0.208	0.124	0.0	6.86	–	–	0.032	0.006	36.908	23.315	–	–	0.008	0.001
5	39.78043	1.24889	1.166	0.975	0.027	0.04	0.261	0.102	0.826	4.343	–	–	0.061	0.024	18.537	18.539	–	–	0.011	0.002
10	15.04997	0.44476	0.621	1.146	0.091	0.062	0.239	0.088	47.02	19.014	–	–	0.077	0.028	51.611	26.149	–	–	0.018	0.002
15	9.81804	0.28928	2.955	1.34	0.029	0.012	0.284	0.089	21.544	37.916	–	–	0.092	0.061	33.001	42.832	–	–	0.028	0.003
20	7.233	0.19874	2.245	1.949	0.07	0.062	0.332	0.094	22.889	57.432	–	–	0.162	0.079	41.776	47.828	–	–	0.035	0.005
25	5.86461	0.15054	4.658	1.881	0.123	0.078	0.223	0.111	23.942	59.568	–	–	0.237	0.087	51.317	75.254	–	–	0.044	0.005
Mean:			**1.612**		**0.073**		**0.259**		**16.59**		**–**		**0.088**		**31.943**		**–**		**0.02**	

Table B.38: Clustering details with Shuttle Control (normalized)

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	20	2000	98	0.4	1.6E+06	2000	624	0.4	1.3E+07	2000	664	0.4	1.3E+07	2000	806	0.04	0.36	1.3E+07	1.6E+06	1.4E+06
3	20	2000	74	0.4	5.0E+06	2000	598	0.4	3.2E+07	2000	541	0.4	3.6E+07	2000	1066	0.28	0.12	3.4E+07	2.2E+06	2.2E+06
4	20	2000	92	0.4	5.0E+06	2000	718	0.4	4.3E+07	2000	456	0.4	4.1E+07	2000	686	0.24	0.16	4.1E+07	3.5E+06	3.0E+06
5	20	2000	74	0.4	8.4E+06	2000	716	0.4	5.8E+07	2000	468	0.4	5.4E+07	2000	786	0.187	0.213	5.8E+07	5.9E+06	4.6E+06
10	20	2000	74	0.4	1.3E+07	2000	718	0.4	1.0E+08	2000	676	0.4	1.0E+08	2000	595	0.16	0.24	1.0E+08	1.1E+07	1.2E+07
15	20	2000	70	0.4	2.6E+07	2000	358	0.4	1.5E+08	2000	436	0.4	1.4E+08	2000	493	0.32	0.08	1.5E+08	1.8E+07	2.0E+07
20	20	2000	62	0.4	3.7E+07	2000	295	0.4	1.8E+08	2000	310	0.4	1.7E+08	2000	414	0.253	0.147	1.7E+08	2.8E+07	2.6E+07
25	20	2000	50	0.4	4.3E+07	2000	255	0.4	1.9E+08	2000	249	0.4	1.9E+08	2000	237	0.08	0.32	1.9E+08	5.2E+07	3.3E+07

B.20 EEG Eye State

Dimensions: $m = 14980$, $n = 14$.

Description: the data set consists of 14 electroencephalogram (EEG) values for predicting the corresponding eye state.

Table B.39: Summary of the results with EEG Eye State ($\times 10^8$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7845.09934	8178.13658	4.245	4.728	0.661	0.358	0.719	0.359	4.247	0.002	0.411	0.21	0.706	0.374	4.246	0.002	0.183	0.199	0.995	0.37
3	1833.88058	1833.87892	0.0	0.003	0.486	0.355	0.638	0.347	0.0	0.003	0.246	0.254	0.855	0.379	0.0	0.003	0.42	0.313	0.755	0.375
4	2.23605	2.23431	0.0	0.001	0.629	0.383	0.678	0.428	0.002	0.001	0.352	0.307	0.563	0.474	0.0	0.001	0.206	0.276	0.615	0.363
5	1.33858	1.33703	-0.0	14.651	0.669	0.35	0.508	0.339	-0.0	120196.81	0.276	0.354	0.583	0.433	-0.0	0.0	0.1	0.221	0.668	0.481
10	0.4531	0.4527	0.001	0.554	0.679	0.363	0.865	0.366	-0.004	88058.848	0.612	0.347	1.088	0.469	-0.005	0.005	0.397	0.307	0.95	0.406
15	0.34653	0.34837	0.622	0.502	0.032	0.015	0.498	0.425	0.055	0.143	0.113	0.037	1.079	0.324	0.135	0.126	0.111	0.19	0.857	0.351
20	0.28986	0.29175	0.785	0.717	0.064	0.367	0.98	0.345	0.02	0.133	0.216	0.055	1.089	0.321	0.06	0.205	0.193	0.063	0.887	0.39
25	0.25989	0.26088	0.636	0.604	0.204	0.274	1.099	0.377	0.156	0.095	0.222	0.049	0.87	0.353	0.137	0.082	0.225	0.042	0.874	0.303
Mean:			**0.786**		**0.428**		**0.748**		**0.559**		**0.306**		**0.854**		**0.571**		**0.229**		**0.825**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7845.09934	8178.13658	4.247	3.973	0.179	0.249	0.842	0.435	-0.0	1.274	–	–	0.003	0.0	-0.0	16.348	–	–	0.004	0.001
3	1833.88058	1833.87892	0.0	0.003	0.245	0.27	0.773	0.42	227.909	98.687	–	–	0.004	0.001	227.909	49.672	–	–	0.005	0.001
4	2.23605	2.23431	0.0	0.001	0.151	0.218	0.798	0.466	268809.803	133731.189	–	–	0.021	0.009	268809.803	128214.104	–	–	0.019	0.007
5	1.33858	1.33703	-0.0	6.519	0.161	0.213	1.077	0.422	449091.754	223405.448	–	–	0.029	0.006	449091.754	205786.243	–	–	0.028	0.005
10	0.4531	0.4527	-0.002	0.006	0.561	0.31	0.96	0.32	1326681.022	632723.737	–	–	0.074	0.035	1326681.023	607938.794	–	–	0.079	0.019
15	0.34653	0.34837	0.058	0.097	0.103	0.051	1.077	0.347	1734685.672	751140.757	–	–	0.197	0.06	1.077	849818.798	–	–	0.145	0.054
20	0.28986	0.29175	0.025	0.031	0.192	0.055	1.105	0.372	2073832.95	989155.199	–	–	0.375	0.117	2073833.29	989155.218	–	–	0.448	0.143
25	0.25989	0.26088	0.109	0.13	0.202	0.07	1.211	0.312	1156493.228	1156492.007	–	–	0.371	0.117	2312984.49	1059942.517	–	–	0.358	0.088
Mean:			**0.555**		**0.224**		**0.98**		**876227.792**		–		**0.134**		**803953.668**		–		**0.136**	

Table B.40: Clustering details with EEG Eye State

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	20	14979	414	1.5	5.8E+07	14979	2898	1.5	4.0E+08	14979	4582	1.5	4.0E+08	14979	4036	0.7	0.8	4.2E+08	1.5E+05	1.8E+05
3	20	14979	368	1.5	8.7E+07	14979	3046	1.5	4.9E+08	14979	2518	1.5	4.8E+08	14979	2674	0.05	1.45	5.0E+08	3.6E+05	4.0E+05
4	20	14979	342	1.5	9.9E+07	14979	1678	1.5	5.7E+08	14979	1809	1.5	5.6E+08	14979	2296	0.3	1.2	5.5E+08	1.5E+06	1.7E+06
5	20	14979	266	1.5	1.2E+08	14979	1426	1.5	5.7E+08	14979	1598	1.5	5.8E+08	14979	2723	0.85	0.65	5.8E+08	3.3E+06	3.0E+06
10	20	14979	308	1.5	1.5E+08	14979	1563	1.5	6.9E+08	14979	1400	1.5	7.0E+08	14979	1357	0.85	0.65	6.9E+08	9.2E+06	1.0E+07
15	20	14979	174	1.5	2.6E+08	14979	935	1.5	7.2E+08	14979	742	1.5	7.0E+08	14979	996	0.35	1.15	7.3E+08	2.4E+07	1.9E+07
20	20	14979	298	1.5	3.3E+08	14979	678	1.5	7.5E+08	14979	558	1.5	7.2E+08	14979	753	1.4	0.1	7.4E+08	3.5E+07	4.3E+07
25	20	14979	286	1.5	3.7E+08	14979	370	1.5	7.3E+08	14979	408	1.5	7.4E+08	14979	490	0.2	1.3	6.7E+08	5.0E+07	4.8E+07

B.21 EEG Eye State (normalized)

Dimensions: $m = 14980$, $n = 14$.

Description: the data set consists of 14 electroencephalogram (EEG) values for predicting the corresponding eye state. Min-max scaling was used for normalization of data set values for better clusterization.

Table B.41: Summary of the results with EEG Eye State (normalized) ($\times 10^1$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	1.15267	1.15216	6.104	9.638	0.258	0.307	0.574	0.311	0.002	8.681	0.193	0.199	0.581	0.287	0.002	0.001	0.139	0.152	0.599	0.274
3	0.82423	0.87097	5.716	13.325	0.005	0.278	0.551	0.267	0.001	9.655	0.009	0.005	0.331	0.29	0.001	1.026	0.009	0.008	0.482	0.278
4	0.5429	0.57038	5.15	14.019	0.005	0.192	0.507	0.297	0.001	10.29	0.012	0.001	0.438	0.259	0.001	0.001	0.011	0.001	0.549	0.301
5	0.28952	0.28903	0.002	33.997	0.413	0.331	0.504	0.315	0.002	15.033	0.161	0.163	0.339	0.255	0.002	0.0	0.195	0.186	0.472	0.281
10	0.10269	0.10335	0.707	0.479	0.029	0.19	0.601	0.303	-0.003	67.68	0.064	0.015	0.449	0.294	-0.004	0.126	0.059	0.014	0.671	0.29
15	0.07469	0.07479	0.2	0.789	0.05	0.24	0.606	0.254	0.036	0.053	0.134	0.049	0.712	0.209	0.052	0.066	0.139	0.045	0.495	0.276
20	0.06125	0.06154	0.457	0.629	0.059	0.077	0.566	0.313	0.177	0.146	0.166	0.06	0.654	0.222	0.205	0.167	0.166	0.054	0.623	0.211
25	0.05385	0.0543	0.873	0.774	0.065	0.186	0.575	0.267	-0.154	0.152	0.224	0.048	0.777	0.192	-0.151	80.458	0.201	0.044	0.637	0.196
Mean:			**2.401**		**0.11**		**0.56**		**0.008**		**0.12**		**0.535**		**0.014**		**0.115**		**0.566**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	1.15267	1.15216	0.001	2.823	0.258	0.256	0.578	0.275	25.398	0.011	–	–	0.016	0.003	25.398	0.0	–	–	0.018	0.003
3	0.82423	0.87097	0.001	7.024	0.009	0.001	0.435	0.25	69.038	0.035	–	–	0.017	0.004	69.038	4.986	–	–	0.015	0.003
4	0.5429	0.57038	0.001	6.519	0.011	0.001	0.56	0.248	152.474	0.048	–	–	0.022	0.007	152.479	0.049	–	–	0.02	0.007
5	0.28952	0.28903	0.002	13.447	0.295	0.235	0.561	0.315	367.097	32.271	–	–	0.033	0.011	367.097	24.212	–	–	0.036	0.009
10	0.10269	0.10335	-0.004	0.131	0.063	0.051	0.789	0.265	633.846	193.064	–	–	0.116	0.037	879.525	132.438	–	–	0.098	0.038
15	0.07469	0.07479	0.037	0.068	0.138	0.146	0.688	0.194	853.035	256.91	–	–	0.179	0.105	853.015	297.335	–	–	0.154	0.062
20	0.06125	0.06154	0.226	0.146	0.176	0.034	0.781	0.239	1044.241	312.789	–	–	0.301	0.151	1044.477	285.929	–	–	0.242	0.092
25	0.05385	0.0543	-0.122	0.197	0.21	0.047	0.925	0.236	1190.906	385.599	–	–	0.44	0.156	1190.787	365.14	–	–	0.303	0.119
Mean:			**0.018**		**0.145**		**0.665**		**542.004**		**–**		**0.141**		**572.727**		**–**		**0.111**	

Table B.42: Clustering details with EEG Eye State (normalized)

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	30	14979	337	1.0	3.6E+07	14979	2470	1.0	2.6E+08	14979	2653	1.0	2.6E+08	14979	2379	0.233	0.767	2.6E+08	9.4E+05	1.0E+06
3	30	14979	314	1.0	5.2E+07	14979	1104	1.0	3.1E+08	14979	1702	1.0	3.1E+08	14979	1378	0.533	0.467	3.0E+08	1.3E+06	1.5E+06
4	30	14979	252	1.0	7.2E+07	14979	1234	1.0	3.5E+08	14979	1438	1.0	3.5E+08	14979	1590	0.933	0.067	3.4E+08	2.3E+06	2.0E+06
5	30	14979	224	1.0	8.3E+07	14979	770	1.0	3.7E+08	14979	1136	1.0	3.8E+08	14979	1370	0.733	0.267	3.7E+08	3.7E+06	3.9E+06
10	30	14979	192	1.0	1.2E+08	14979	560	1.0	4.3E+08	14979	862	1.0	4.3E+08	14979	1040	0.5	0.5	4.3E+08	1.4E+07	1.3E+07
15	30	14979	166	1.0	1.7E+08	14979	602	1.0	4.6E+08	14979	358	1.0	4.5E+08	14979	531	0.333	0.667	4.7E+08	2.0E+07	2.0E+07
20	30	14979	152	1.0	2.0E+08	14979	361	1.0	4.6E+08	14979	318	1.0	4.6E+08	14979	442	0.733	0.267	4.7E+08	3.1E+07	3.3E+07
25	30	14979	174	1.0	2.5E+08	14979	282	1.0	4.6E+08	14979	228	1.0	4.7E+08	14979	415	0.933	0.067	4.7E+08	4.3E+07	4.1E+07

B.22 Pla85900

Dimensions: $m = 85900$, $n = 2$.

Description: a data set contains cities coordinates for traveling salesman problem.

Table B.43: Summary of the results with Pla85900 ($\times 10^{15}$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.74908	0.60031	0.054	0.718	0.59	0.405	0.776	0.44	0.013	0.021	0.19	0.276	0.603	0.429	0.011	0.224	0.108	0.17	0.73	0.44
3	2.28057	0.36407	0.026	0.036	0.809	0.508	0.941	0.444	0.024	0.021	0.259	0.307	0.552	0.428	0.025	0.032	0.274	0.233	0.735	0.391
5	1.33972	0.21512	0.09	0.751	0.292	0.34	0.821	0.438	0.046	0.029	0.099	0.083	0.683	0.42	0.051	0.305	0.051	0.331	0.718	0.427
10	0.68294	0.10944	0.587	0.371	0.85	0.459	0.802	0.468	0.111	0.148	0.221	0.297	0.923	0.48	0.151	0.317	0.146	0.274	0.844	0.433
15	0.46029	0.07355	0.291	0.476	0.557	0.504	0.919	0.433	0.251	0.153	0.539	0.349	0.769	0.36	0.268	0.183	0.335	0.398	0.979	0.438
20	0.34988	0.05595	0.656	0.422	0.545	0.379	0.833	0.421	0.316	0.254	0.507	0.36	0.971	0.411	0.304	0.338	0.3	0.191	0.655	0.38
25	0.28259	0.04518	0.89	0.318	0.853	0.383	0.884	0.373	0.617	0.281	0.618	0.392	0.806	0.408	0.763	0.455	0.432	0.304	0.711	0.402
Mean:			**0.371**		**0.642**		**0.854**		**0.197**		**0.348**		**0.758**		**0.225**		**0.235**		**0.767**	

k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\overline{t}$		t		ε		$\overline{t}$		t		ε		$\overline{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.74908	0.60031	0.016	0.014	0.139	0.224	0.988	0.428	0.0	0.686	–	–	0.024	0.007	6.458	2.618	–	–	0.01	0.002
3	2.28057	0.36407	0.024	0.029	0.287	0.353	0.671	0.41	0.0	0.0	–	–	0.078	0.025	0.005	20.855	–	–	0.022	0.006
5	1.33972	0.21512	0.036	0.027	0.091	0.089	1.106	0.456	0.407	1.133	–	–	0.082	0.06	6.719	5.906	–	–	0.027	0.012
10	0.68294	0.10944	0.136	0.186	0.273	0.318	1.022	0.348	0.42	0.774	–	–	0.201	0.087	14.084	10.514	–	–	0.067	0.017
15	0.46029	0.07355	0.226	0.143	0.555	0.281	1.004	0.337	0.495	0.806	–	–	0.313	0.156	17.409	9.89	–	–	0.098	0.022
20	0.34988	0.05595	0.331	0.126	0.469	0.305	0.728	0.394	0.45	0.601	–	–	0.453	0.213	15.152	8.883	–	–	0.125	0.043
25	0.28259	0.04518	0.618	0.299	0.645	0.359	0.996	0.369	0.932	0.495	–	–	0.697	0.229	13.672	7.007	–	–	0.163	0.038
Mean:			**0.198**		**0.351**		**0.931**		**0.386**		–		**0.264**		**10.5**		–		**0.073**	

Table B.44: Clustering details with Pla85900

k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	40	14000	237	1.5	4.4E+07	14000	1470	1.5	3.4E+08	14000	1718	1.5	3.2E+08	14000	2442	1.0	0.5	3.4E+08	5.1E+06	4.1E+06
3	40	14000	240	1.5	8.1E+07	14000	1190	1.5	5.9E+08	14000	1476	1.5	6.0E+08	14000	1413	1.4	0.1	6.0E+08	1.6E+07	1.1E+07
5	40	14000	217	1.5	1.4E+08	14000	1238	1.5	9.3E+08	14000	1272	1.5	9.1E+08	14000	1943	1.05	0.45	9.1E+08	1.9E+07	1.6E+07
10	40	14000	186	1.5	3.7E+08	14000	1000	1.5	1.7E+09	14000	907	1.5	1.7E+09	14000	1098	1.2	0.3	1.7E+09	7.6E+07	5.8E+07
15	40	14000	159	1.5	5.6E+08	14000	498	1.5	2.4E+09	14000	748	1.5	2.3E+09	14000	728	0.6	0.9	2.3E+09	1.4E+08	1.0E+08
20	40	14000	117	1.5	7.6E+08	14000	482	1.5	2.7E+09	14000	336	1.5	2.7E+09	14000	359	1.4	0.1	2.7E+09	2.2E+08	1.4E+08
25	40	14000	110	1.5	9.3E+08	14000	270	1.5	2.8E+09	14000	264	1.5	2.9E+09	14000	324	1.15	0.35	2.8E+09	3.5E+08	1.9E+08

B.23 D15112

Dimensions: $m = 15112$, $n = 2$.

Description: a data set with German cities coordinates for travelling salesman problem.

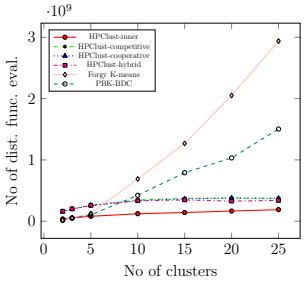
Table B.45: Summary of the results with D15112 ($\times 10^{11}$)

k	f^*	$\overline{f}$	HPClust-inner						HPClust-competitive						HPClust-cooperative					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.68403	1.91227	0.011	0.012	0.84	0.495	0.84	0.44	0.014	0.014	0.218	0.268	0.602	0.222	0.02	0.023	0.258	0.231	0.745	0.256
3	2.5324	1.30699	0.021	0.023	0.289	0.54	1.009	0.425	0.023	0.019	0.277	0.238	0.45	0.299	0.036	0.027	0.205	0.166	0.619	0.379
5	1.32707	0.68683	0.041	0.023	0.507	0.402	0.907	0.427	0.034	0.022	0.07	0.123	0.801	0.399	0.045	0.02	0.178	0.159	0.777	0.329
10	0.64491	0.33574	0.734	1.319	0.495	0.411	0.649	0.458	0.118	0.145	0.104	0.205	0.973	0.434	0.098	0.278	0.15	0.356	1.158	0.315
15	0.43136	0.22393	0.776	0.79	0.205	0.127	0.546	0.389	0.235	0.091	0.247	0.194	0.365	0.369	0.309	0.2	0.163	0.323	0.596	0.444
20	0.32177	0.16878	0.888	0.619	0.214	0.171	0.558	0.449	0.28	0.144	0.098	0.081	1.023	0.398	0.626	0.497	0.063	0.041	0.623	0.4
25	0.25308	0.13159	0.868	0.851	0.396	0.516	0.623	0.409	0.306	0.206	0.487	0.257	0.626	0.338	0.867	0.432	0.675	0.342	0.945	0.361
Mean:			**0.477**		**0.421**		**0.733**		**0.144**		**0.214**		**0.691**		**0.286**		**0.242**		**0.78**	

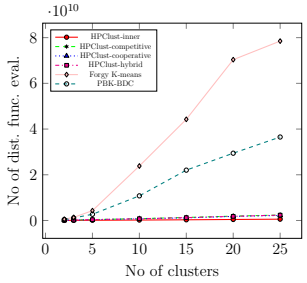
k	f^*	$\overline{f}$	HPClust-hybrid						Forgy K-means						PBK-BDC					
			ε		$\bar{t}$		t		ε		$\bar{t}$		t		ε		$\bar{t}$		t	
			med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.68403	1.91227	0.021	0.012	0.293	0.276	0.909	0.428	0.0	0.0	–	–	0.003	0.0	0.013	0.008	–	–	0.003	0.0
3	2.5324	1.30699	0.03	0.04	0.342	0.24	0.799	0.341	0.001	0.0	–	–	0.007	0.002	0.038	0.084	–	–	0.004	0.001
5	1.32707	0.68683	0.052	0.029	0.254	0.22	1.129	0.346	-0.0	7.357	–	–	0.005	0.002	0.048	4.148	–	–	0.004	0.001
10	0.64491	0.33574	0.1	0.033	0.126	0.214	1.1	0.296	1.411	1.559	–	–	0.032	0.02	0.955	1.46	–	–	0.018	0.006
15	0.43136	0.22393	0.283	0.147	0.381	0.372	0.663	0.467	2.788	1.452	–	–	0.045	0.013	2.639	1.792	–	–	0.015	0.005
20	0.32177	0.16878	0.3	0.155	0.071	0.049	0.818	0.377	1.635	2.513	–	–	0.05	0.014	3.321	2.902	–	–	0.019	0.006
25	0.25308	0.13159	0.297	0.339	0.226	0.293	0.91	0.432	2.208	1.762	–	–	0.093	0.037	2.838	1.386	–	–	0.04	0.012
Mean:			**0.155**		**0.242**		**0.904**		**1.149**		**–**		**0.033**		**1.407**		**–**		**0.015**	

Table B.46: Clustering details with D15112

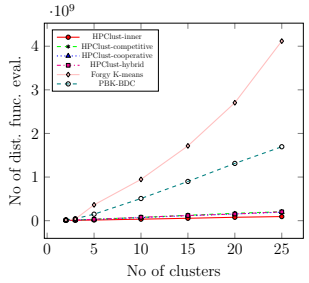
k	n_{exec}	HPClust-inner				HPClust-competitive				HPClust-cooperative				HPClust-hybrid					Forgy K-means	PBK-BDC
		s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T	n_d	s	n_s	T_1	T_2	n_d	n_d	n_d
2	15	8000	1083	1.5	8.7E+07	8000	6286	1.5	4.6E+08	8000	7270	1.5	4.7E+08	8000	9539	0.95	0.55	7.3E+08	4.8E+05	2.4E+05
3	15	8000	1184	1.5	1.5E+08	8000	3347	1.5	8.5E+08	8000	4976	1.5	8.3E+08	8000	6275	0.7	0.8	1.1E+09	1.9E+06	9.3E+05
5	15	8000	759	1.5	2.0E+08	8000	5774	1.5	1.3E+09	8000	5444	1.5	1.4E+09	8000	7672	0.7	0.8	1.4E+09	1.5E+06	9.2E+05
10	15	8000	392	1.5	4.5E+08	8000	3398	1.5	2.3E+09	8000	3959	1.5	2.3E+09	8000	3865	0.9	0.6	2.3E+09	8.9E+06	3.8E+06
15	15	8000	304	1.5	6.2E+08	8000	622	1.5	2.7E+09	8000	1352	1.5	2.9E+09	8000	1290	1.35	0.15	2.6E+09	1.5E+07	6.9E+06
20	15	8000	231	1.5	8.9E+08	8000	1400	1.5	2.9E+09	8000	1015	1.5	3.1E+09	8000	1087	1.35	0.15	3.0E+09	2.5E+07	9.6E+06
25	15	8000	207	1.5	9.7E+08	8000	615	1.5	3.0E+09	8000	1099	1.5	3.1E+09	8000	964	0.85	0.65	3.1E+09	2.6E+07	1.3E+07



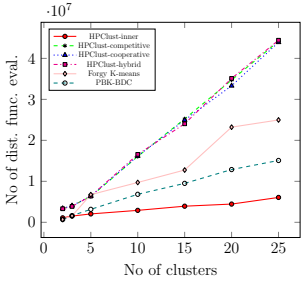
(a) CORD-19 Embeddings



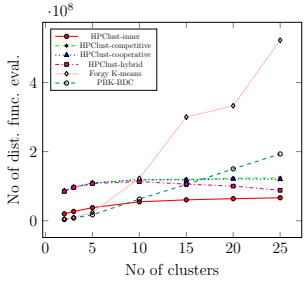
(b) HEPMASS



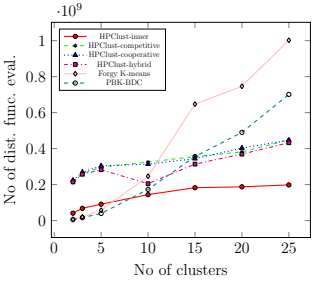
(c) US Census Data 1990



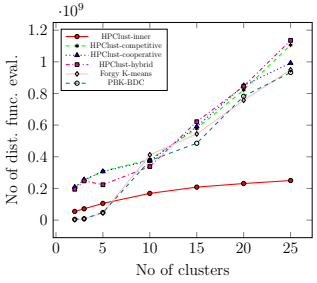
(d) Gisette



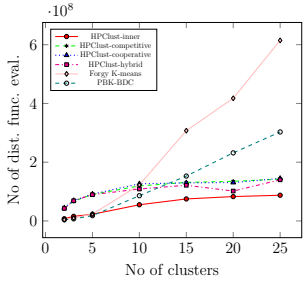
(e) Music Analysis



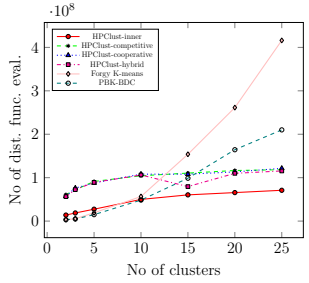
(f) Protein Homology



(g) MiniBooNE Particle Identification

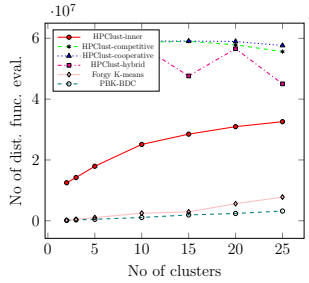


(h) MiniBooNE Particle Identification (normalized)

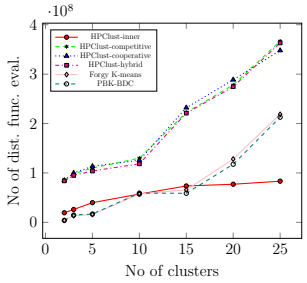


(i) MFCCs for Speech Emotion Recognition

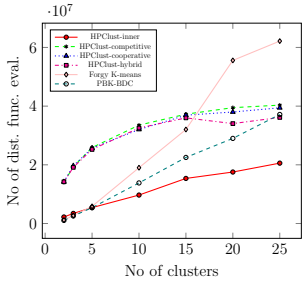
Figure B.1: Distance function evaluations. Set 1



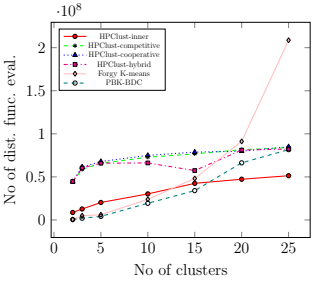
(a) ISOLET



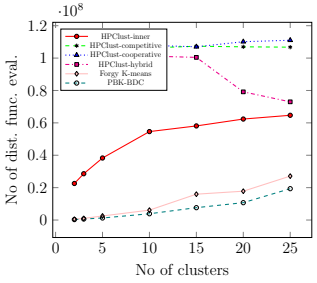
(b) Sensorless Drive Diagnosis



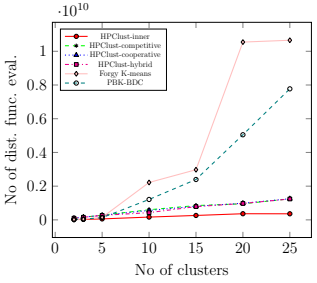
(c) Sensorless Drive Diagnosis (normalized)



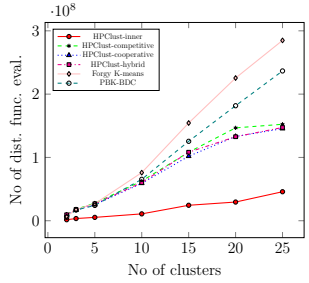
(d) Online News Popularity



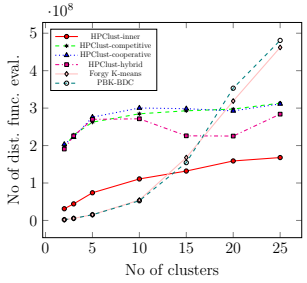
(e) Gas Sensor Array Drift



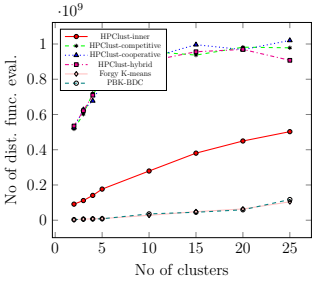
(f) 3D Road Network



(g) Skin Segmentation

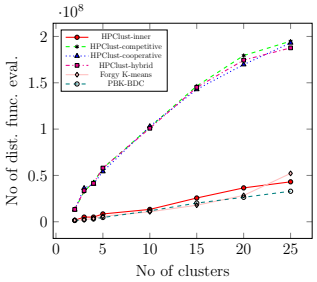


(h) KEGG Metabolic Relation Network (Directed)

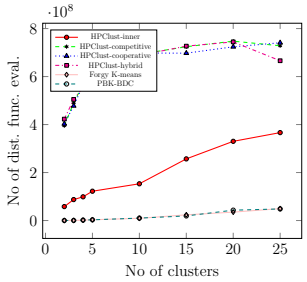


(i) Shuttle Control

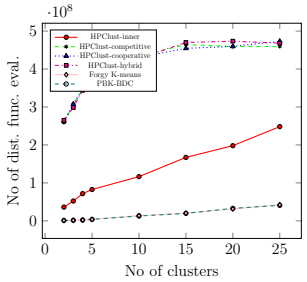
Figure B.2: Distance function evaluations. Set 2



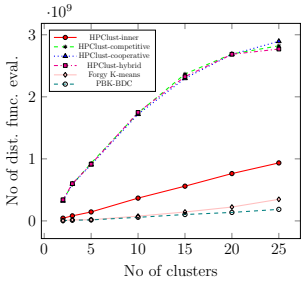
(a) Shuttle Control (normalized)



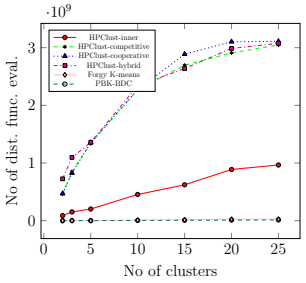
(b) EEG Eye State



(c) EEG Eye State (normalized)



(d) Pla85900



(e) D15112

Figure B.3: Distance function evaluations. Set 3

Appendix C

COMPARISON OF BIG DATA CLUSTERING ALGORITHMS

C.1 CORD-19 Embeddings

Table C.1: Summary of the results with CORD-19 Embeddings ($\times 10^9, m = 599616, n = 768$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	2.03893*	0.01	0.0	27.73	9.28	2.2	3.8	6.79	2.7	0.0	0.0	2.33	0.16	0.0	0.01	10.63	3.82
3	1.9093*	0.01	0.0	21.5	10.04	0.92	2.81	11.11	0.37	0.06	2.25	10.08	1.57	0.07	0.66	9.61	1.53
5	1.77676*	0.02	0.06	12.39	11.69	2.48	1.2	15.88	0.39	2.26	3.52	15.64	3.0	0.86	1.15	26.38	7.26
10	1.62555*	0.04	0.05	27.17	7.29	1.76	1.12	28.32	0.72	4.17	1.7	60.12	11.24	0.94	0.48	6.13	1.27
15	1.55295*	0.12	0.07	34.61	4.6	1.89	0.43	41.53	0.62	1.62	0.55	102.28	11.81	1.2	0.29	17.94	1.87
20	1.49987*	0.23	0.09	29.21	6.57	2.31	0.45	54.1	0.49	2.88	1.3	152.44	34.08	1.37	0.3	29.46	8.28
25	1.46394*	0.12	0.12	31.3	7.94	2.07	0.5	66.86	0.57	1.92	0.79	181.09	17.81	1.26	0.32	9.75	1.26
Mean:		**0.08**		**26.27**		**1.95**		**32.08**		**1.84**		**74.85**		**0.81**		**15.7**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	2.03893*	0.0	0.0	19.99	2.07	19.78	0.08	38.17	17.75	0.0	0.0	15.96	6.54	0.01	0.0	10.01	5.94
3	1.9093*	0.01	1.99	55.72	16.15	27.29	0.76	27.68	7.65	-0.0	0.0	49.67	1.85	0.02	0.01	12.57	3.9
5	1.77676*	0.13	1.09	139.77	41.48	33.96	2.34	16.99	0.69	0.13	1.08	86.02	49.91	0.02	0.34	15.3	1.89
10	1.62555*	0.43	0.21	675.22	259.77	33.34	6.75	23.42	0.64	0.02	0.26	533.11	289.38	0.15	0.23	26.38	8.46
15	1.55295*	0.23	0.57	1072.21	438.69	35.74	9.6	30.67	1.04	0.52	0.32	1084.2	770.22	0.58	0.46	51.16	17.82
20	1.49987*	0.12	0.2	1167.53	737.09	23.87	9.32	40.98	8.34	0.29	0.15	1265.77	565.41	0.43	0.3	43.91	15.6
25	1.46394*	-0.01	0.08	2577.51	813.09	24.06	6.6	88.2	4.27	0.01	0.27	1822.84	878.27	0.54	0.34	74.97	12.38
Mean:		**0.13**		**815.42**		**28.29**		**38.02**		**0.14**		**693.94**		**0.25**		**33.47**	

Table C.2: Clustering details with CORD-19 Embeddings

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	7	32000	591	37.33	2.67	1.7E+08	4.8E+06	1.2E+07	42	2.7E+06	1.4E+07	32000	7.1E+07	32000	62.5	16	0.5	1.0E+09	32000	3.0E+06
3	7	32000	381	32.0	8.0	2.2E+08	7.9E+06	5.1E+07	68	6.5E+06	5.6E+07	32000	4.2E+07	32000	62.5	16	0.5	1.1E+09	32000	5.8E+06
5	7	32000	142	21.33	18.67	2.8E+08	1.3E+07	1.1E+08	81	1.3E+07	1.6E+08	32000	3.6E+07	32000	62.5	16	0.5	1.1E+09	32000	9.8E+06
10	7	32000	184	24.0	16.0	3.6E+08	2.5E+07	4.4E+08	71	2.3E+07	8.6E+08	32000	5.3E+07	32000	62.5	16	0.5	1.7E+09	32000	2.3E+07
15	7	32000	110	26.67	13.33	3.7E+08	3.7E+07	7.3E+08	73	3.5E+07	1.4E+09	32000	7.6E+07	32000	62.5	16	0.5	2.5E+09	32000	6.7E+07
20	7	32000	42	8.0	32.0	3.5E+08	4.9E+07	1.0E+09	86	5.5E+07	1.5E+09	32000	9.8E+07	32000	62.5	16	0.5	2.9E+09	32000	4.5E+07
25	7	32000	24	32.0	8.0	3.7E+08	6.2E+07	1.4E+09	75	6.0E+07	3.7E+09	32000	1.2E+08	32000	62.5	16	0.5	3.6E+09	32000	9.9E+07

C.2 HEPMASS

Table C.3: Summary of the results with HEPMASS ($\times 10^8, m = 10500000, n = 27$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	2.48889*	0.0	0.0	12.72	7.67	0.08	2.52	24.35	2.1	0.0	0.0	2.91	0.28	0.01	0.03	9.73	2.21
3	2.36789*	0.01	0.01	12.73	7.17	1.93	0.49	41.72	1.48	0.38	0.79	5.25	0.48	1.29	0.58	10.96	1.57
5	2.21106*	0.33	0.16	22.27	5.92	1.88	0.66	61.15	2.16	3.1	2.43	9.07	0.81	1.26	0.76	14.15	2.19
10	2.00353*	0.02	0.05	23.13	5.73	3.13	0.79	112.1	3.44	1.26	1.23	27.19	1.88	1.2	1.09	12.84	2.58
15	1.89922*	0.18	0.08	19.67	6.86	2.46	0.38	143.8	1.8	1.63	0.77	51.17	1.16	1.12	0.52	8.23	0.72
20	1.82904*	0.17	0.06	24.68	10.35	2.81	0.71	183.13	4.07	2.53	0.54	70.66	5.14	1.21	0.43	20.9	4.47
25	1.77524*	0.13	0.08	23.64	10.68	3.37	0.56	221.85	3.29	2.4	0.66	78.79	3.16	1.3	0.35	10.62	0.75
Mean:		**0.12**		**19.84**		**2.24**		**112.59**		**1.61**		**35.01**		**1.06**		**12.49**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	2.48889*	0.0	0.0	31.37	20.57	13.0	0.63	28.88	3.87	0.0	0.0	13.83	3.86	0.0	0.0	5.59	0.21
3	2.36789*	0.0	0.7	39.93	20.92	17.13	1.1	22.64	6.19	0.0	0.56	25.57	12.63	1.25	0.56	5.79	0.16
5	2.21106*	0.32	0.19	90.69	23.6	22.48	1.36	24.59	2.9	0.32	0.36	56.38	8.13	0.82	0.26	5.55	0.28
10	2.00353*	0.28	0.3	431.3	198.04	29.31	2.31	35.22	2.15	0.42	0.38	237.68	82.07	0.9	0.38	7.33	1.76
15	1.89922*	0.33	0.13	770.54	220.21	32.29	2.8	45.82	1.26	0.18	0.12	562.15	456.89	0.31	0.49	7.49	1.59
20	1.82904*	0.26	0.14	1386.78	552.29	34.49	1.47	61.31	2.58	0.07	0.18	820.05	1400.54	0.25	0.34	8.8	0.63
25	1.77524*	0.25	0.23	1383.96	473.33	35.52	1.88	127.2	3.64	0.23	0.23	1173.44	520.57	0.31	0.24	10.01	1.23
Mean:		**0.21**		**590.65**		**26.32**		**49.38**		**0.17**		**412.73**		**0.55**		**7.22**	

Table C.4: Clustering details with HEPMASS

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	7	64000	108	17.0	13.0	1.1E+08	9.7E+07	4.1E+08	332	4.2E+07	7.1E+08	64000	1.9E+08	64000	5.0	16	0.5	4.6E+09	64000	4.4E+07
3	7	64000	92	7.0	23.0	2.0E+08	1.5E+08	1.2E+09	394	7.6E+07	1.5E+09	64000	1.4E+08	64000	5.0	16	0.5	5.1E+09	64000	6.0E+07
5	7	64000	213	7.0	23.0	3.7E+08	2.4E+08	2.7E+09	515	1.6E+08	4.2E+09	64000	1.4E+08	64000	5.0	16	0.5	6.9E+09	64000	8.6E+07
10	7	64000	186	16.0	14.0	8.0E+08	4.8E+08	1.1E+10	525	3.4E+08	2.4E+10	64000	2.1E+08	64000	5.0	16	0.5	1.8E+10	64000	1.9E+08
15	7	64000	160	9.0	21.0	1.4E+09	7.2E+08	2.2E+10	572	5.5E+08	4.4E+10	64000	2.9E+08	64000	5.0	16	0.5	3.8E+10	64000	2.9E+08
20	7	64000	225	28.0	2.0	2.0E+09	9.6E+08	2.9E+10	546	7.0E+08	8.3E+10	64000	3.8E+08	64000	5.0	16	0.5	5.5E+10	64000	4.0E+08
25	7	64000	196	22.0	8.0	2.6E+09	1.2E+09	3.7E+10	511	8.2E+08	8.9E+10	64000	4.7E+08	64000	5.0	16	0.5	8.3E+10	64000	5.2E+08

C.3 US Census Data 1990

Table C.5: Summary of the results with US Census Data 1990 ($\times 10^8, m = 2458285, n = 68$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	18.39812*	0.2	0.14	2.33	0.84	0.0	120.22	4.69	0.16	0.0	0.0	0.37	0.02	0.0	0.01	0.57	0.37
3	6.1591*	0.08	0.04	1.9	0.92	162.97	59.74	6.87	0.38	170.1	43.26	0.65	0.02	0.0	0.0	0.6	0.41
5	3.35214*	0.13	0.03	1.85	0.7	357.16	173.52	13.82	1.71	393.91	168.91	1.56	0.12	0.02	2.68	2.22	1.15
10	2.36352*	1.96	2.01	2.85	0.69	17.01	183.68	25.1	0.97	21.65	204.93	3.5	0.15	4.19	3.21	1.91	0.84
15	2.04097*	2.03	0.96	2.56	0.62	18.3	131.91	36.61	1.18	18.31	190.83	5.64	0.23	3.91	1.4	1.8	0.81
20	1.81278*	2.14	1.15	1.81	0.86	16.58	7.27	48.33	0.94	17.2	6.57	7.88	0.68	3.96	1.9	2.32	0.85
25	1.64602*	2.14	0.96	2.8	0.88	17.34	7.63	62.95	1.4	15.42	8.2	11.54	0.71	4.06	1.45	3.08	1.15
Mean:		**1.24**		**2.3**		**84.2**		**28.34**		**90.94**		**4.45**		**2.31**		**1.78**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	18.39812*	0.0	0.0	4.23	0.21	1.74	0.97	2.54	0.11	0.0	0.0	0.84	0.3	0.03	0.04	3.68	0.12
3	6.1591*	0.0	80.36	6.39	0.93	2.55	0.33	2.62	0.1	0.0	0.0	0.88	0.19	0.04	59.35	3.66	0.24
5	3.35214*	4.14	76.79	15.81	6.59	17.26	2.82	3.19	0.46	16.02	25.24	6.05	6.89	78.97	90.73	3.38	0.87
10	2.36352*	7.53	3.45	54.0	16.78	38.82	12.91	4.95	0.25	10.59	11.32	27.84	18.88	138.83	112.26	3.39	0.14
15	2.04097*	7.27	4.26	83.82	29.54	35.16	5.86	5.86	0.48	11.82	3.36	60.96	48.11	171.1	68.72	3.3	0.27
20	1.81278*	7.05	3.82	139.92	32.13	42.25	4.36	3.74	0.74	13.93	3.49	108.2	100.25	203.75	77.55	3.08	0.98
25	1.64602*	8.66	4.16	209.28	58.04	36.64	6.21	2.98	0.3	14.76	4.07	178.2	51.25	235.02	72.35	2.88	0.55
Mean:		**4.95**		**73.35**		**24.92**		**3.7**		**9.59**		**54.71**		**118.25**		**3.34**	

Table C.6: Clustering details with US Census Data 1990

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	20	6000	219	0.2	2.8	1.3E+07	2.0E+07	1.9E+07	18	2.2E+05	1.5E+07	6000	2.2E+07	6000	4.5	16	0.5	4.6E+07	6000	9.9E+06
3	20	6000	188	2.1	0.9	1.9E+07	2.9E+07	4.4E+07	34	6.0E+05	2.2E+07	6000	2.2E+07	6000	4.5	16	0.5	5.1E+07	6000	1.2E+07
5	20	6000	194	0.6	2.4	3.5E+07	5.6E+07	1.5E+08	350	1.0E+07	1.6E+08	6000	2.6E+07	6000	4.5	16	0.5	1.5E+08	6000	1.7E+07
10	20	6000	238	2.4	0.6	8.8E+07	1.1E+08	5.1E+08	306	1.8E+07	9.1E+08	6000	4.0E+07	6000	4.5	16	0.5	6.5E+08	6000	3.0E+07
15	20	6000	186	1.9	1.1	1.3E+08	1.6E+08	9.2E+08	300	2.7E+07	1.3E+09	6000	5.3E+07	6000	4.5	16	0.5	1.5E+09	6000	4.3E+07
20	20	6000	66	0.1	2.9	1.8E+08	2.1E+08	1.3E+09	422	5.1E+07	2.7E+09	6000	6.4E+07	6000	4.5	16	0.5	2.5E+09	6000	5.6E+07
25	20	6000	124	2.5	0.5	2.2E+08	2.6E+08	1.7E+09	433	6.5E+07	4.0E+09	6000	7.6E+07	6000	4.5	16	0.5	4.0E+09	6000	6.8E+07

C.4 Gisette

Table C.7: Summary of the results with Gisette ($\times 10^{12}, m = 13500, n = 5000$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	4.19944	0.01	0.01	3.2	0.73	0.62	0.41	1.03	0.03	0.01	0.0	2.51	0.67	0.18	0.33	2.97	0.9
3	4.11596	0.02	0.03	6.61	5.76	0.74	0.3	1.53	0.09	0.01	0.0	9.41	2.82	0.26	0.23	6.66	2.41
5	4.02303	0.06	0.05	7.24	1.25	0.64	0.26	2.5	0.18	0.04	0.04	14.6	9.06	0.34	0.14	8.37	2.4
10	3.87672	0.15	0.05	17.71	2.27	0.84	0.32	5.23	0.21	0.07	0.06	28.14	14.24	0.54	0.16	5.83	2.13
15	3.81766	-0.31	0.05	25.4	3.22	0.62	0.27	10.31	0.63	-0.37	0.05	47.56	13.94	0.04	0.16	6.72	1.66
20	3.81436	-1.64	0.05	32.27	2.63	-0.86	0.29	11.7	0.24	-1.67	0.05	66.77	30.02	-1.24	0.13	11.01	5.81
25	3.74937	-1.05	0.07	44.76	4.2	-0.22	0.33	15.34	0.6	-1.01	0.08	72.81	16.45	-0.66	0.1	4.46	1.04
Mean:		**-0.39**		**19.6**		**0.34**		**6.8**		**-0.42**		**34.54**		**-0.08**		**6.57**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	4.19944	0.0	0.0	4.98	1.61	4.12	0.13	11.12	0.28	-0.0	0.0	13.71	1.7	0.02	0.0	5.1	0.94
3	4.11596	0.0	0.0	11.81	3.56	5.92	0.19	11.4	1.98	0.0	0.0	15.85	2.65	0.04	0.01	11.95	3.95
5	4.02303	0.01	0.05	26.7	7.79	7.96	0.4	14.5	10.47	0.01	0.02	54.1	24.21	0.09	0.04	18.54	4.57
10	3.87672	0.04	0.07	58.23	20.81	11.03	0.42	31.14	1.42	0.07	0.07	88.15	27.77	0.23	0.07	36.59	15.63
15	3.81766	-0.45	0.04	92.51	32.23	12.02	0.18	38.75	4.8	-0.46	0.03	87.11	22.65	-0.16	0.08	45.05	12.38
20	3.81436	-1.76	0.04	115.48	26.47	11.89	0.22	39.23	1.36	-1.78	0.05	139.05	29.58	-1.44	0.09	52.36	11.42
25	3.74937	-1.17	0.04	133.57	42.05	13.62	0.25	30.97	0.64	-1.21	0.07	179.89	54.19	-0.7	0.15	68.2	12.7
Mean:		**-0.48**		**63.33**		**9.51**		**25.3**		**-0.48**		**82.55**		**-0.27**		**33.97**	

Table C.8: Clustering details with Gisette

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	15	10000	9	4.5	0.5	3.7E+06	1.1E+05	4.5E+05	26	5.2E+05	7.0E+05	10000	5.5E+06	10000	22000.0	16	0.5	1.0E+08	10000	5.1E+05
3	15	10000	9	1.83	3.17	4.2E+06	1.6E+05	1.8E+06	24	7.2E+05	1.8E+06	10000	5.5E+06	10000	22000.0	16	0.5	1.0E+08	10000	1.8E+06
5	15	10000	6	3.17	1.83	6.3E+06	2.7E+05	2.9E+06	31	1.6E+06	4.6E+06	10000	7.9E+06	10000	22000.0	16	0.5	1.1E+08	10000	2.8E+06
10	15	10000	6	3.83	1.17	1.6E+07	5.4E+05	5.8E+06	26	2.6E+06	1.1E+07	10000	1.5E+07	10000	22000.0	16	0.5	1.2E+08	10000	6.4E+06
15	15	10000	5	3.0	2.0	2.6E+07	8.1E+05	9.7E+06	30	4.5E+06	1.7E+07	10000	2.3E+07	10000	22000.0	16	0.5	1.2E+08	10000	7.4E+06
20	15	10000	6	2.33	2.67	3.4E+07	1.1E+06	1.4E+07	28	5.6E+06	2.2E+07	10000	2.6E+07	10000	22000.0	16	0.5	1.3E+08	10000	9.3E+06
25	15	10000	7	4.67	0.33	4.3E+07	1.4E+06	1.5E+07	30	7.5E+06	2.4E+07	10000	2.1E+07	10000	22000.0	16	0.5	1.3E+08	10000	1.2E+07

C.5 Music Analysis

Table C.9: Summary of the results with Music Analysis ($\times 10^{11}, m = 106574, n = 518$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	5.00474*	0.08	0.03	4.94	1.89	0.81	2.88	0.84	0.03	0.42	0.99	0.35	0.04	0.35	5.59	0.89	0.24
3	3.83748*	0.16	0.06	4.71	1.92	11.57	6.72	1.19	0.01	1.96	15.02	0.89	0.27	0.76	5.54	0.52	0.11
5	2.74249*	0.24	0.72	4.55	1.96	13.33	9.42	2.05	0.11	4.98	19.55	1.7	0.34	2.79	3.36	0.77	0.68
10	1.87296*	0.56	0.65	4.58	2.31	10.45	3.55	3.51	0.07	6.1	6.31	5.69	1.63	3.41	1.69	3.72	1.69
15	1.54422*	1.1	0.36	4.5	2.18	15.3	4.28	5.21	0.05	6.15	4.85	9.86	1.54	3.32	1.25	1.52	0.69
20	1.35315*	1.66	0.48	6.56	2.11	13.44	4.42	11.43	1.07	5.74	2.91	13.65	2.44	3.14	0.81	0.68	0.13
25	1.22622*	1.99	0.75	7.23	2.27	15.97	3.13	8.5	0.04	5.95	3.17	16.19	2.45	3.27	0.94	1.24	0.42
Mean:		**0.83**		**5.29**		**11.55**		**4.68**		**4.47**		**6.9**		**2.43**		**1.34**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	5.00474*	-0.0	10.4	3.72	0.48	30.96	9.36	3.07	0.18	-0.0	0.0	3.0	0.1	0.04	9.28	1.43	0.55
3	3.83748*	-0.0	4.15	5.23	4.12	52.97	9.72	2.71	0.09	9.06	1.97	9.82	1.22	0.09	4.33	1.37	0.11
5	2.74249*	1.12	1.4	16.08	3.78	39.73	22.57	3.16	0.26	2.23	1.42	19.0	4.32	2.39	1.85	1.74	0.22
10	1.87296*	1.56	2.04	61.44	25.53	42.8	17.53	4.08	1.13	1.67	0.5	60.01	12.01	1.62	2.68	2.91	0.82
15	1.54422*	0.29	0.52	106.37	43.67	40.91	11.44	4.34	0.1	0.65	0.36	153.38	38.67	2.68	1.96	3.89	0.8
20	1.35315*	0.18	0.7	161.72	104.7	46.31	5.99	4.01	0.42	1.04	0.44	177.15	68.8	2.78	2.12	5.39	1.99
25	1.22622*	0.74	0.51	201.65	80.42	56.47	13.0	4.45	0.9	0.7	0.54	272.22	77.43	3.57	1.28	6.85	2.06
Mean:		**0.55**		**79.46**		**44.31**		**3.69**		**2.19**		**99.23**		**1.88**		**3.37**	

Table C.10: Clustering details with Music Analysis

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	20	6000	1264	1.33	6.67	9.0E+07	8.5E+05	3.5E+06	38	4.5E+05	4.9E+06	6000	1.0E+07	6000	500.0	16	0.5	4.1E+07	6000	6.3E+05
3	20	6000	656	1.6	6.4	1.0E+08	1.3E+06	8.4E+06	33	5.9E+05	7.5E+06	6000	8.9E+06	6000	500.0	16	0.5	5.4E+07	6000	8.7E+05
5	20	6000	336	1.33	6.67	1.1E+08	2.1E+06	1.7E+07	45	1.4E+06	2.9E+07	6000	8.6E+06	6000	500.0	16	0.5	7.3E+07	6000	1.9E+06
10	20	6000	85	6.13	1.87	1.2E+08	4.3E+06	6.1E+07	43	2.6E+06	1.2E+08	6000	1.1E+07	6000	500.0	16	0.5	1.6E+08	6000	4.8E+06
15	20	6000	36	4.53	3.47	1.2E+08	6.4E+06	1.1E+08	43	3.9E+06	2.1E+08	6000	1.3E+07	6000	500.0	16	0.5	3.6E+08	6000	7.3E+06
20	20	6000	36	0.53	7.47	1.1E+08	8.5E+06	1.5E+08	43	5.2E+06	3.2E+08	6000	1.2E+07	6000	500.0	16	0.5	4.0E+08	6000	1.1E+07
25	20	6000	22	0.27	7.73	1.2E+08	1.1E+07	2.0E+08	32	4.8E+06	4.0E+08	6000	1.1E+07	6000	500.0	16	0.5	6.1E+08	6000	1.5E+07

C.6 Protein Homology

Table C.11: Summary of the results with Protein Homology ($\times 10^{11}, m = 145751, n = 74$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	15.20433*	1.87	0.01	1.3	1.05	2.65	0.93	0.26	0.55	1.83	0.0	0.33	0.05	2.26	2.62	0.57	0.19
3	8.07129*	0.79	0.37	3.1	1.0	66.78	9.6	0.39	0.1	0.02	53.0	0.9	0.2	66.34	5.0	0.59	0.14
5	5.30537*	0.93	0.6	2.12	0.83	130.11	13.43	0.6	0.14	19.35	9.58	1.88	0.33	118.23	9.23	0.77	0.2
10	3.3767*	0.19	0.19	3.34	0.63	223.7	38.77	1.28	0.09	32.58	6.99	5.12	1.63	165.18	42.26	0.93	0.34
15	2.86473*	0.99	0.37	3.22	1.12	263.65	50.83	1.66	0.28	31.35	5.8	9.95	1.77	115.05	66.0	0.97	0.28
20	2.5732*	1.16	0.41	4.45	1.78	262.05	72.07	2.17	0.13	34.3	11.71	15.6	3.87	170.19	88.61	1.15	0.29
25	2.38539*	0.87	0.47	4.34	0.82	239.88	85.73	2.77	0.07	33.53	13.46	21.4	5.09	70.04	50.31	1.24	0.23
Mean:		**0.97**		**3.12**		**169.83**		**1.3**		**21.85**		**7.88**		**101.04**		**0.89**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	15.20433*	0.0	0.89	0.21	0.91	63.07	29.67	233.4	17.83	1.82	0.0	3.12	0.43	0.0	0.0	0.3	4.47
3	8.07129*	0.0	31.4	0.74	0.33	91.91	43.08	104.94	6.41	0.0	0.0	4.37	0.14	0.01	0.0	0.37	0.03
5	5.30537*	0.0	0.19	2.47	0.5	160.61	12.34	57.45	3.71	0.0	0.0	6.37	0.32	0.02	0.01	0.95	0.22
10	3.3767*	-0.0	7.15	9.49	2.16	282.51	13.15	50.3	1.84	18.12	0.0	17.29	1.63	18.17	6.32	3.31	0.88
15	2.86473*	0.21	0.95	20.55	7.25	333.95	15.32	56.5	2.45	23.94	0.03	38.66	7.86	24.04	8.04	6.42	1.5
20	2.5732*	0.96	0.46	32.45	7.61	370.47	9.61	71.5	1.66	28.56	0.18	49.28	5.73	28.36	13.44	8.65	3.79
25	2.38539*	1.62	0.74	34.93	14.68	401.85	26.31	82.36	2.69	31.85	0.16	66.93	16.05	32.14	14.23	10.44	1.9
Mean:		**0.4**		**14.41**		**243.48**		**93.78**		**14.9**		**26.57**		**14.68**		**4.35**	

Table C.12: Clustering details with Protein Homology

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	15	56000	230	3.27	0.23	2.3E+08	1.2E+06	5.4E+06	19	2.1E+06	2.0E+06	56000	3.7E+08	56000	1000.0	16	0.5	3.1E+09	56000	1.0E+06
3	15	56000	403	2.57	0.93	2.6E+08	1.7E+06	1.8E+07	27	4.5E+06	1.1E+07	56000	2.5E+08	56000	1000.0	16	0.5	3.2E+09	56000	4.1E+06
5	15	56000	126	0.93	2.57	2.9E+08	2.9E+06	4.2E+07	24	6.7E+06	4.1E+07	56000	1.7E+08	56000	1000.0	16	0.5	3.2E+09	56000	1.6E+07
10	15	56000	22	0.23	3.27	2.2E+08	5.8E+06	1.7E+08	28	1.6E+07	1.7E+08	56000	1.5E+08	56000	1000.0	16	0.5	3.4E+09	56000	6.2E+07
15	15	56000	5	0.23	3.27	2.9E+08	8.7E+06	3.2E+08	30	2.5E+07	4.1E+08	56000	1.7E+08	56000	1000.0	16	0.5	3.8E+09	56000	1.3E+08
20	15	56000	6	1.17	2.33	3.9E+08	1.2E+07	4.9E+08	28	3.1E+07	6.1E+08	56000	1.9E+08	56000	1000.0	16	0.5	4.0E+09	56000	1.6E+08
25	15	56000	6	3.15	0.35	4.5E+08	1.5E+07	7.1E+08	37	5.2E+07	6.6E+08	56000	2.1E+08	56000	1000.0	16	0.5	4.4E+09	56000	2.1E+08

C.7 MiniBooNE Particle Identification

Table C.13: Summary of the results with MiniBooNE Particle Identification ($\times 10^{10}, m = 130064, n = 50$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	8.92236	0.0	0.0	1.08	0.95	286908.11	284.03	0.19	0.0	0.0	143134.9	0.17	0.09	286911.79	8.34	1.17	0.45
3	5.22601	0.0	0.0	2.0	0.81	489832.22	195901.07	0.27	0.05	0.0	122192.54	0.32	0.04	489836.85	279.94	0.59	0.55
5	1.82252	0.01	0.0	2.36	0.63	127.27	561815.57	0.43	0.12	0.0	58.26	1.21	0.45	1404170.45	1381.36	1.08	0.5
10	0.9092	0.03	0.01	4.48	0.98	43.32	92.24	0.83	0.21	0.0	0.41	8.5	3.28	2812016.23	5271.38	1.05	0.79
15	0.63506	0.15	0.56	6.07	1.62	80.4	45.66	1.19	0.15	3.88	0.78	17.39	4.57	4026475.67	7873.09	1.96	1.5
20	0.50863	0.1	0.19	5.9	1.53	87.89	30.75	1.61	0.12	7.05	0.59	23.62	6.11	5018494.92	10938.68	0.9	0.14
25	0.44425	-0.24	1437586.3	6.94	1.58	80.34	28.55	1.95	0.35	8.93	0.33	34.44	6.15	5746462.36	23826.26	1.1	0.24
Mean:		**0.01**		**4.12**		**111022.79**		**0.93**		**2.84**		**12.24**		**2826338.32**		**1.12**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	8.92236	0.0	0.0	0.09	0.0	286928.91	0.13	426.12	19.95	286920.93	126881.21	0.7	0.09	0.0	0.0	0.09	0.02
3	5.22601	0.0	5.41	0.33	0.11	489868.34	1.68	310.33	18.45	21.68	0.0	0.9	0.06	0.02	8.67	0.15	0.02
5	1.82252	0.0	39.7	1.07	0.37	1404812.42	14.0	249.76	19.01	0.0	0.0	1.93	0.13	0.11	48.17	0.3	0.07
10	0.9092	0.0	2.6	7.38	3.83	2816035.19	10.77	205.28	12.66	0.0	0.0	15.09	0.64	1.99	3.26	1.33	0.44
15	0.63506	2.32	1.61	10.35	5.52	4031654.69	13.36	183.52	11.52	3.88	0.59	20.44	2.15	4.61	2.04	3.0	1.05
20	0.50863	0.97	3.36	16.24	5.54	5033808.44	8.95	165.96	12.4	7.05	0.55	28.46	5.26	9.88	2.41	3.1	0.69
25	0.44425	0.04	4.21	20.29	7.22	5763295.53	11.78	164.5	10.23	8.94	0.2	39.09	6.08	12.18	3.28	4.77	1.68
Mean:		**0.48**		**7.96**		**2832343.36**		**243.64**		**40994.64**		**15.23**		**4.11**		**1.82**	

Table C.14: Clustering details with MiniBooNE Particle Identification

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	15	130000	100	2.5	0.5	1.5E+08	1.0E+06	2.9E+06	32	8.3E+06	7.8E+05	40000	4.4E+08	40000	70.0	16	0.5	1.6E+09	40000	7.6E+05
3	15	130000	170	2.5	0.5	2.4E+08	1.6E+06	8.6E+06	15	5.8E+06	7.0E+06	40000	3.8E+08	40000	70.0	16	0.5	1.6E+09	40000	2.3E+06
5	15	130000	73	0.2	2.8	2.5E+08	2.6E+06	3.6E+07	23	1.5E+07	2.9E+07	40000	3.3E+08	40000	70.0	16	0.5	1.6E+09	40000	6.7E+06
10	15	130000	14	2.0	1.0	3.5E+08	5.2E+06	2.6E+08	19	2.5E+07	2.1E+08	40000	3.0E+08	40000	70.0	16	0.5	2.0E+09	40000	3.9E+07
15	15	130000	9	0.5	2.5	6.1E+08	7.8E+06	5.5E+08	19	3.7E+07	2.9E+08	40000	3.1E+08	40000	70.0	16	0.5	2.2E+09	40000	8.9E+07
20	15	130000	5	2.9	0.1	7.9E+08	1.0E+07	7.4E+08	19	4.9E+07	5.0E+08	40000	3.2E+08	40000	70.0	16	0.5	2.4E+09	40000	9.6E+07
25	15	130000	4	1.7	1.3	1.1E+09	1.3E+07	1.1E+09	26	8.4E+07	6.4E+08	40000	3.3E+08	40000	70.0	16	0.5	2.8E+09	40000	1.5E+08

C.8 MiniBooNE Particle Identification (normalized)

Table C.15: Summary of the results with MiniBooNE Particle Identification (normalized) ($\times 10^2, m = 130064, n = 50$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	28.01938*	0.02	0.01	0.43	0.25	684.56	38.55	0.57	0.14	4.17	1864.82	0.06	0.02	686.34	26.05	0.14	0.04
3	19.85673*	0.03	0.02	0.68	0.23	580.41	468.8	0.76	0.13	9.47	21.74	0.08	0.02	947.47	151.64	0.18	0.09
5	12.10267*	0.09	0.05	0.76	0.21	-0.0	7.46	1.48	0.24	13.95	16.86	0.21	0.04	1625.43	703.04	0.4	0.35
10	8.57382*	0.56	0.76	0.68	0.25	3.12	1.17	4.89	0.88	8.42	5.08	0.87	0.19	3.77	85.05	0.28	0.13
15	7.24131*	0.78	0.35	0.79	0.25	2.66	1.75	7.67	1.07	8.68	4.79	1.86	0.35	3.4	1.25	0.3	0.06
20	6.30493*	1.33	0.52	1.1	0.33	2.98	2.23	11.2	0.87	10.07	4.49	2.4	0.49	4.13	1.26	0.3	0.05
25	5.71335*	1.35	0.51	1.18	0.41	2.32	0.75	15.74	1.29	8.14	3.58	2.92	0.49	4.74	1.27	0.48	0.1
Mean:		**0.59**		**0.8**		**182.29**		**6.04**		**8.99**		**1.2**		**467.9**		**0.3**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	28.01938*	0.0	149.2	0.08	0.06	3.09	319.04	4.0	0.2	0.0	0.0	0.15	0.01	0.01	0.01	0.12	0.01
3	19.85673*	3.49	3.49	0.48	0.11	19.17	6.86	3.97	0.48	-0.0	0.0	0.53	0.06	7.02	3.34	0.14	0.02
5	12.10267*	-0.0	2.67	0.95	0.41	40.53	17.34	4.44	0.18	-0.0	0.83	1.16	0.17	0.14	1.91	0.16	0.03
10	8.57382*	0.31	0.98	4.45	2.1	37.64	14.94	6.72	0.44	2.48	0.93	6.0	1.03	1.15	1.43	0.4	0.1
15	7.24131*	0.28	0.83	8.47	3.67	43.43	7.36	8.59	0.33	0.54	1.63	11.12	5.71	1.3	0.76	0.54	0.13
20	6.30493*	0.83	0.57	15.36	6.38	43.98	7.18	12.33	0.58	0.6	0.61	15.32	4.37	2.62	1.23	0.61	0.22
25	5.71335*	0.23	0.24	18.57	9.43	49.78	8.67	9.26	0.49	0.29	0.16	23.9	6.86	3.51	1.21	0.85	0.21
Mean:		**0.73**		**6.91**		**33.94**		**7.04**		**0.56**		**8.31**		**2.25**		**0.4**	

Table C.16: Clustering details with MiniBooNE Particle Identification (normalized)

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	20	12000	418	0.03	0.97	5.0E+07	1.9E+06	4.1E+06	14	3.5E+05	7.8E+05	12000	4.0E+07	12000	0.03	16	0.5	1.4E+08	12000	5.9E+05
3	20	12000	451	0.03	0.97	7.8E+07	2.9E+06	7.2E+06	22	7.9E+05	8.2E+06	12000	3.3E+07	12000	0.03	16	0.5	1.5E+08	12000	1.2E+06
5	20	12000	298	0.17	0.83	1.0E+08	5.9E+06	1.8E+07	74	4.4E+06	2.3E+07	12000	3.0E+07	12000	0.03	16	0.5	1.7E+08	12000	2.6E+06
10	20	12000	50	0.67	0.33	1.1E+08	1.9E+07	7.9E+07	47	5.6E+06	1.2E+08	12000	3.3E+07	12000	0.03	16	0.5	3.0E+08	12000	7.0E+06
15	20	12000	24	0.87	0.13	1.3E+08	3.6E+07	1.6E+08	42	7.5E+06	2.4E+08	12000	3.9E+07	12000	0.03	16	0.5	4.4E+08	12000	1.6E+07
20	20	12000	12	0.23	0.77	1.1E+08	5.4E+07	2.3E+08	47	1.1E+07	4.7E+08	12000	4.5E+07	12000	0.03	16	0.5	6.0E+08	12000	1.8E+07
25	20	12000	10	0.73	0.27	1.4E+08	7.2E+07	3.1E+08	44	1.3E+07	5.7E+08	12000	4.1E+07	12000	0.03	16	0.5	8.9E+08	12000	2.7E+07

C.9 MFCCs for Speech Emotion Recognition

Table C.17: Summary of the results with MFCCs for Speech Emotion Recognition ($\times 10^9, m = 85134, n = 58$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	0.74513*	0.04	0.02	0.67	0.29	0.77	4.71	0.13	0.01	0.0	0.0	0.05	0.01	0.13	2.48	0.16	0.07
3	0.50215*	0.04	0.05	0.38	0.23	3.93	8.01	0.18	0.03	69.81	36.42	0.07	0.01	0.2	4.53	0.23	0.08
5	0.3456*	0.05	0.03	0.47	0.22	10.13	5.12	0.29	0.04	25.73	27.26	0.19	0.04	2.1	3.28	0.25	0.07
10	0.21763*	0.13	0.04	0.7	0.24	13.59	7.17	0.57	0.07	8.23	5.86	0.7	0.14	3.89	2.45	0.35	0.07
15	0.17608*	0.47	0.65	0.85	0.25	14.35	4.47	0.82	0.13	8.91	4.99	1.07	0.22	3.29	1.27	0.29	0.05
20	0.15383*	0.8	0.49	0.8	0.32	19.41	6.93	1.3	0.15	13.61	4.85	2.07	0.31	3.55	1.38	0.38	0.07
25	0.14109*	0.96	0.5	1.29	0.48	15.31	4.71	1.66	0.3	13.04	4.11	2.68	0.33	3.32	0.79	0.35	0.07
Mean:		**0.36**		**0.74**		**11.07**		**0.71**		**19.9**		**0.98**		**2.35**		**0.29**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	0.74513*	0.0	0.0	0.22	0.05	17.73	6.96	7.12	0.62	0.0	0.0	0.35	0.02	0.02	0.01	0.11	0.02
3	0.50215*	0.0	0.0	0.3	0.05	29.78	24.69	6.24	0.33	0.0	0.0	0.42	0.06	0.03	0.02	0.1	0.02
5	0.3456*	-0.0	0.0	0.75	0.17	32.44	15.24	6.89	0.51	-0.0	0.0	1.21	0.25	0.05	0.05	0.2	0.05
10	0.21763*	1.88	1.25	2.25	0.79	62.49	16.29	7.98	0.77	-0.01	1.17	2.83	0.67	2.71	2.2	0.32	0.14
15	0.17608*	1.35	0.77	4.5	2.06	49.94	11.76	10.29	0.76	1.7	1.58	6.19	2.56	3.47	1.64	0.5	0.13
20	0.15383*	1.14	1.31	7.44	2.49	44.23	13.8	14.43	1.27	2.1	1.36	10.78	2.7	3.12	2.01	0.76	0.35
25	0.14109*	0.67	1.23	11.75	3.21	39.19	6.69	10.01	0.72	4.94	1.79	16.19	5.31	2.87	2.07	1.07	0.34
Mean:		**0.72**		**3.89**		**39.4**		**8.99**		**1.25**		**5.42**		**1.75**		**0.44**	

Table C.18: Clustering details with MFCCs for Speech Emotion Recognition

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	20	12000	612	0.77	0.23	6.6E+07	6.8E+05	3.3E+06	25	6.0E+05	3.9E+06	12000	6.2E+07	12000	25.0	16	0.5	1.5E+08	12000	8.0E+05
3	20	12000	226	0.2	0.8	8.1E+07	1.0E+06	4.8E+06	36	1.3E+06	5.6E+06	12000	5.2E+07	12000	25.0	16	0.5	1.5E+08	12000	1.1E+06
5	20	12000	144	0.83	0.17	9.6E+07	1.7E+06	1.5E+07	36	2.2E+06	1.6E+07	12000	4.4E+07	12000	25.0	16	0.5	1.6E+08	12000	2.5E+06
10	20	12000	58	0.97	0.03	1.1E+08	3.4E+06	4.8E+07	37	4.4E+06	5.4E+07	12000	4.0E+07	12000	25.0	16	0.5	2.1E+08	12000	7.3E+06
15	20	12000	14	0.03	0.97	8.0E+07	5.1E+06	9.0E+07	42	7.6E+06	1.2E+08	12000	4.2E+07	12000	25.0	16	0.5	3.0E+08	12000	1.3E+07
20	20	12000	12	0.9	0.1	1.1E+08	7.9E+06	1.8E+08	40	9.5E+06	1.9E+08	12000	4.6E+07	12000	25.0	16	0.5	4.2E+08	12000	2.0E+07
25	20	12000	14	0.83	0.17	1.3E+08	1.0E+07	2.1E+08	32	9.4E+06	3.0E+08	12000	4.3E+07	12000	25.0	16	0.5	5.6E+08	12000	2.9E+07

C.10 ISOLET

Table C.19: Summary of the results with ISOLET ($\times 10^5, m = 7797, n = 617$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7.2194	0.03	0.01	1.65	1.15	0.09	4.02	0.07	0.01	0.02	0.01	0.05	0.02	0.03	0.07	0.18	0.04
3	6.78782	0.04	0.01	3.33	1.03	1.17	0.94	0.11	0.01	0.59	0.26	0.14	0.05	0.6	0.62	0.21	0.07
5	6.13651	0.07	0.14	2.58	1.25	2.83	1.4	0.17	0.02	0.47	0.56	0.35	0.2	1.92	1.21	0.28	0.06
10	5.28577	0.31	0.12	2.99	1.28	3.15	1.86	0.32	0.01	0.47	1.44	0.57	0.2	2.14	1.1	0.38	0.06
15	4.87391	0.81	0.38	4.0	1.06	3.94	1.18	0.55	0.14	1.82	1.02	0.84	0.4	2.93	0.86	0.39	0.1
20	4.60857	0.66	0.45	4.41	1.11	4.04	1.41	0.64	0.01	1.96	0.83	1.08	0.37	2.47	0.68	0.34	0.07
25	4.44323	0.19	0.26	4.2	1.2	3.54	1.13	0.79	0.01	1.8	0.62	1.35	0.59	2.17	0.81	0.35	0.06
Mean:		**0.3**		**3.31**		**2.68**		**0.38**		**1.02**		**0.63**		**1.75**		**0.3**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7.2194	-0.0	0.0	0.22	0.06	19.2	3.27	1.32	0.06	-0.0	0.0	0.24	0.03	0.05	0.02	0.18	0.02
3	6.78782	0.55	0.52	0.49	0.39	24.29	4.43	1.45	0.08	0.55	0.26	0.5	0.39	0.13	0.53	0.22	0.08
5	6.13651	0.39	0.49	0.78	0.18	24.07	7.12	1.72	0.11	0.39	0.48	1.01	0.17	0.56	1.15	0.39	0.15
10	5.28577	1.06	1.17	1.36	0.78	50.87	12.37	2.79	0.16	1.39	0.68	2.12	0.85	2.34	1.32	0.78	0.27
15	4.87391	0.82	0.46	2.18	1.04	57.14	6.7	2.55	0.11	2.23	0.87	2.11	0.73	2.14	1.24	1.08	0.48
20	4.60857	0.93	0.94	3.52	1.28	54.25	4.81	2.07	0.11	2.07	0.69	3.81	1.19	2.64	1.32	1.18	0.39
25	4.44323	0.66	0.56	4.92	1.52	43.09	6.57	1.69	0.06	2.76	1.08	4.68	1.42	2.29	0.94	1.52	0.29
Mean:		**0.63**		**1.92**		**38.99**		**1.94**		**1.34**		**2.07**		**1.45**		**0.77**	

Table C.20: Clustering details with ISOLET

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	15	4000	622	1.17	3.83	5.1E+07	6.3E+04	1.0E+05	20	1.6E+05	1.9E+05	4000	2.5E+06	4000	9.0	16	0.5	1.6E+07	4000	1.2E+05
3	15	4000	936	1.67	3.33	5.5E+07	1.0E+05	2.8E+05	22	2.6E+05	5.1E+05	4000	2.3E+06	4000	9.0	16	0.5	1.6E+07	4000	2.8E+05
5	15	4000	419	1.83	3.17	5.8E+07	1.7E+05	6.0E+05	33	6.6E+05	1.1E+06	4000	2.9E+06	4000	9.0	16	0.5	1.7E+07	4000	5.9E+05
10	15	4000	143	4.67	0.33	6.0E+07	3.3E+05	1.1E+06	33	1.3E+06	1.9E+06	4000	5.3E+06	4000	9.0	16	0.5	1.9E+07	4000	1.2E+06
15	15	4000	101	1.17	3.83	5.3E+07	5.0E+05	1.5E+06	29	1.7E+06	3.0E+06	4000	4.7E+06	4000	9.0	16	0.5	1.9E+07	4000	1.8E+06
20	15	4000	81	3.33	1.67	5.4E+07	6.8E+05	2.2E+06	33	2.6E+06	4.8E+06	4000	3.8E+06	4000	9.0	16	0.5	2.2E+07	4000	2.1E+06
25	15	4000	47	3.17	1.83	5.3E+07	8.5E+05	2.7E+06	27	2.7E+06	7.2E+06	4000	3.2E+06	4000	9.0	16	0.5	2.4E+07	4000	2.7E+06

C.11 Sensorless Drive Diagnosis

Table C.21: Summary of the results with Sensorless Drive Diagnosis ($\times 10^7, m = 58509, n = 48$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.88116	-0.0	2.25	0.68	0.29	100.86	1.12	0.11	0.03	100.19	0.0	0.14	0.05	102.98	1.85	0.25	0.09
3	2.91313	-0.0	0.55	0.69	0.24	159.87	3.01	0.16	0.03	155.19	72.12	0.31	0.15	162.74	3.9	0.36	0.11
5	1.93651	0.02	1.05	0.78	0.23	273.99	6.38	0.25	0.01	37.86	35.66	0.46	0.08	271.65	5.39	0.34	0.12
10	0.98472	-2.4	0.93	1.39	0.46	596.17	121.42	0.47	0.05	127.2	0.04	1.8	0.4	589.72	6.23	0.45	0.13
15	0.62816	0.02	1.27	1.58	0.59	959.36	260.82	0.69	0.13	235.44	0.51	1.78	0.54	951.77	16.69	0.36	0.08
20	0.49884	-0.08	0.44	2.06	0.67	1221.22	296.82	0.91	0.09	309.27	24.05	3.74	1.28	1208.06	33.43	0.48	0.12
25	0.42225	0.95	2.34	2.65	0.51	1398.7	364.82	1.14	0.02	315.17	12.88	6.61	1.54	1435.83	38.46	0.44	0.07
Mean:		**-0.21**		**1.4**		**672.88**		**0.53**		**182.9**		**2.12**		**674.68**		**0.38**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.88116	-0.0	45.91	0.04	0.1	107.02	4.22	2.43	0.11	100.19	0.0	0.4	0.05	0.01	0.01	0.03	0.0
3	2.91313	10.86	39.12	0.2	0.13	168.74	4.83	2.74	0.1	10.86	71.42	0.87	0.23	10.91	0.03	0.05	0.01
5	1.93651	25.53	16.58	0.43	0.22	289.24	7.4	3.5	0.6	37.86	74.9	1.06	0.14	37.97	8.29	0.07	0.02
10	0.98472	14.54	9.69	1.28	0.3	650.79	9.22	5.07	0.4	127.2	0.04	2.59	0.47	108.26	24.43	0.17	0.06
15	0.62816	17.38	15.85	1.83	0.76	1069.0	10.48	6.74	0.24	235.52	0.45	2.29	0.44	198.12	63.83	0.25	0.09
20	0.49884	3.61	2.59	2.68	0.68	1368.43	9.32	7.37	0.38	309.34	22.34	4.12	1.65	260.43	66.21	0.33	0.08
25	0.42225	3.42	6.86	3.49	1.11	1634.51	12.01	5.93	0.28	315.71	40.71	7.48	1.58	284.68	101.69	0.41	0.1
Mean:		**10.76**		**1.42**		**755.39**		**4.82**		**162.38**		**2.69**		**128.63**		**0.19**	

Table C.22: Clustering details with Sensorless Drive Diagnosis

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	40	58500	237	0.27	0.73	8.9E+07	4.7E+05	4.0E+06	17	2.0E+06	4.7E+05	10000	1.5E+07	10000	4.0	16	0.5	1.0E+08	10000	2.9E+05
3	40	58500	151	0.2	0.8	1.0E+08	7.8E+05	9.8E+06	20	3.5E+06	5.4E+06	10000	1.4E+07	10000	4.0	16	0.5	1.2E+08	10000	9.2E+05
5	40	58500	72	0.83	0.17	1.1E+08	1.3E+06	1.5E+07	17	5.0E+06	1.2E+07	10000	1.4E+07	10000	4.0	16	0.5	1.2E+08	10000	1.7E+06
10	40	58500	12	0.63	0.37	1.2E+08	2.5E+06	6.0E+07	20	1.1E+07	3.9E+07	10000	1.8E+07	10000	4.0	16	0.5	1.6E+08	10000	5.2E+06
15	40	58500	3	0.4	0.6	2.2E+08	3.8E+06	6.1E+07	20	1.8E+07	5.6E+07	10000	2.2E+07	10000	4.0	16	0.5	1.6E+08	10000	8.3E+06
20	40	58500	4	0.53	0.47	2.7E+08	5.0E+06	1.2E+08	18	2.0E+07	8.5E+07	10000	2.3E+07	10000	4.0	16	0.5	2.2E+08	10000	1.1E+07
25	40	58500	5	0.77	0.23	3.5E+08	6.3E+06	2.2E+08	18	2.6E+07	1.1E+08	10000	2.1E+07	10000	4.0	16	0.5	3.4E+08	10000	1.4E+07

C.12 Sensorless Drive Diagnosis (normalized)

Table C.23: Summary of the results with Sensorless Drive Diagnosis (normalized) ($\times 10^3, m = 58509, n = 48$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	15.64798*	0.1	0.04	0.16	0.09	0.0	16.58	0.11	0.02	0.0	0.88	0.01	0.0	0.03	0.13	0.14	0.05
3	12.19375*	0.11	0.99	0.21	0.09	3.57	3.22	0.18	0.07	1.78	21.07	0.03	0.01	3.6	2.53	0.11	0.06
5	7.85054*	0.31	0.23	0.14	0.07	3.7	7.55	0.29	0.04	4.02	9.62	0.05	0.01	0.82	2.9	0.13	0.05
10	4.71275*	0.65	1.11	0.26	0.08	7.95	7.17	0.68	0.12	12.71	11.38	0.12	0.03	6.1	2.66	0.13	0.04
15	3.62541*	1.34	1.04	0.29	0.08	9.08	6.52	1.15	0.21	16.44	8.86	0.21	0.04	6.43	2.86	0.15	0.03
20	2.971*	2.39	1.49	0.26	0.06	13.94	7.85	1.61	0.13	18.27	7.85	0.27	0.05	6.22	2.74	0.15	0.04
25	2.60929*	2.97	1.44	0.26	0.07	19.58	5.65	2.15	0.24	21.43	6.85	0.31	0.04	5.15	2.18	0.15	0.11
Mean:		**1.12**		**0.23**		**8.26**		**0.88**		**10.66**		**0.14**		**4.05**		**0.14**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	15.64798*	0.0	11.81	0.06	0.02	72.6	28.17	0.91	0.05	0.0	19.93	0.08	0.02	0.05	20.18	0.03	0.0
3	12.19375*	0.98	2.94	0.14	0.06	38.3	39.72	1.37	0.16	0.98	2.15	0.15	0.12	1.05	2.36	0.03	0.0
5	7.85054*	0.5	2.05	0.24	0.15	50.8	33.14	1.3	0.12	0.54	4.73	0.16	0.13	0.81	6.36	0.03	0.0
10	4.71275*	5.51	2.93	0.57	0.34	36.15	18.53	1.9	0.13	6.58	3.65	0.61	0.31	8.87	11.11	0.05	0.01
15	3.62541*	6.07	3.43	1.11	0.45	43.43	24.48	1.63	0.16	8.29	3.7	1.15	0.41	17.39	12.37	0.06	0.02
20	2.971*	7.88	3.68	1.49	0.63	51.39	20.4	1.27	0.09	12.01	4.0	1.61	0.59	26.03	15.3	0.07	0.02
25	2.60929*	8.98	4.14	2.06	0.83	38.42	10.9	1.09	0.09	14.03	5.16	1.69	0.84	26.57	11.61	0.08	0.02
Mean:		**4.27**		**0.81**		**47.3**		**1.35**		**6.06**		**0.78**		**11.54**		**0.05**	

Table C.24: Clustering details with Sensorless Drive Diagnosis (normalized)

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	40	3500	364	0.16	0.14	1.5E+07	5.6E+05	1.1E+06	36	2.5E+05	1.2E+06	3500	8.7E+06	3500	0.12	16	0.5	1.3E+07	3500	3.0E+05
3	40	3500	383	0.13	0.17	2.1E+07	9.3E+05	2.7E+06	34	3.6E+05	3.0E+06	3500	8.8E+06	3500	0.12	16	0.5	1.5E+07	3500	4.2E+05
5	40	3500	169	0.01	0.29	2.7E+07	1.6E+06	5.3E+06	41	7.2E+05	6.7E+06	3500	9.4E+06	3500	0.12	16	0.5	1.6E+07	3500	6.9E+05
10	40	3500	160	0.23	0.07	3.4E+07	3.7E+06	1.4E+07	44	1.5E+06	1.7E+07	3500	1.2E+07	3500	0.12	16	0.5	2.8E+07	3500	1.4E+06
15	40	3500	78	0.26	0.04	3.6E+07	6.2E+06	2.3E+07	46	2.4E+06	3.5E+07	3500	1.1E+07	3500	0.12	16	0.5	4.5E+07	3500	2.2E+06
20	40	3500	37	0.1	0.2	3.6E+07	8.8E+06	2.9E+07	44	3.1E+06	4.6E+07	3500	9.9E+06	3500	0.12	16	0.5	6.0E+07	3500	2.8E+06
25	40	3500	22	0.1	0.2	3.5E+07	1.2E+07	3.6E+07	40	3.5E+06	6.3E+07	3500	9.2E+06	3500	0.12	16	0.5	6.8E+07	3500	3.5E+06

C.13 Online News Popularity

Table C.25: Summary of the results with Online News Popularity ($\times 10^{14}, m = 39644, n = 58$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	9.53913	0.02	0.01	0.36	0.17	54.81	50.35	0.08	0.01	0.0	0.0	0.01	0.0	0.0	0.34	0.11	0.03
3	5.91077	0.05	0.04	0.36	0.14	27.82	53.54	0.08	0.0	0.01	29.47	0.06	0.02	0.13	0.77	0.12	0.04
5	3.09885	0.08	0.02	0.4	0.21	57.38	42.11	0.19	0.04	53.35	38.44	0.11	0.04	0.87	7.9	0.13	0.03
10	1.17247	1.07	1.28	0.56	0.15	109.15	62.97	0.26	0.08	45.13	56.84	0.47	0.18	38.73	22.55	0.2	0.06
15	0.77637	2.4	3.95	0.59	0.2	147.05	59.73	0.39	0.09	46.83	32.05	0.98	0.21	19.9	27.7	0.19	0.04
20	0.59809	3.04	1.24	0.57	0.18	147.28	50.87	0.51	0.11	44.99	9.32	1.86	0.34	23.75	6.54	0.23	0.07
25	0.49616	3.88	2.81	0.63	0.33	127.89	63.92	0.64	0.05	46.05	13.31	2.32	0.51	32.08	3.87	0.28	0.04
Mean:		**1.51**		**0.49**		**95.91**		**0.31**		**33.77**		**0.83**		**16.5**		**0.18**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	9.53913	-0.0	0.0	0.02	0.02	2.03	0.2	5.92	0.43	-0.0	0.0	0.15	0.01	0.01	54.01	0.03	0.0
3	5.91077	0.0	12.85	0.16	0.06	28.56	11.9	5.87	0.44	0.0	12.75	0.33	0.08	34.61	19.8	0.03	0.02
5	3.09885	12.07	15.65	0.15	0.17	32.96	20.18	5.08	0.33	12.07	9.38	0.52	0.1	23.25	30.06	0.06	0.03
10	1.17247	3.69	4.73	0.57	0.49	153.44	37.58	6.12	0.48	49.41	23.4	1.27	0.65	16.23	12.92	0.15	0.06
15	0.77637	9.9	5.16	1.07	0.42	213.96	62.36	7.93	0.62	14.63	2.82	2.7	0.89	14.38	11.13	0.21	0.12
20	0.59809	13.25	6.25	1.51	0.29	194.3	79.4	9.21	0.52	20.87	2.65	3.04	1.15	17.17	9.72	0.38	0.16
25	0.49616	8.03	7.49	2.05	0.86	339.74	78.75	6.95	0.29	27.01	2.38	6.32	2.31	12.01	6.97	0.4	0.21
Mean:		**6.71**		**0.79**		**137.86**		**6.73**		**17.71**		**2.05**		**16.81**		**0.18**	

Table C.26: Clustering details with Online News Popularity

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	20	10000	617	0.63	0.07	5.0E+07	3.2E+05	4.8E+05	19	3.8E+05	3.2E+05	10000	6.4E+07	10000	10000.0	16	0.5	1.0E+08	10000	2.2E+05
3	20	10000	236	0.14	0.56	6.4E+07	4.8E+05	2.0E+06	18	5.6E+05	3.9E+06	10000	5.7E+07	10000	10000.0	16	0.5	1.0E+08	10000	4.1E+05
5	20	10000	186	0.47	0.23	6.7E+07	7.9E+05	4.1E+06	21	1.0E+06	3.1E+06	10000	4.8E+07	10000	10000.0	16	0.5	1.1E+08	10000	1.2E+06
10	20	10000	87	0.49	0.21	7.1E+07	1.6E+06	1.6E+07	30	3.0E+06	1.4E+07	10000	4.3E+07	10000	10000.0	16	0.5	1.2E+08	10000	3.8E+06
15	20	10000	14	0.05	0.65	6.2E+07	2.4E+06	3.5E+07	22	3.3E+06	2.8E+07	10000	4.3E+07	10000	10000.0	16	0.5	1.7E+08	10000	5.8E+06
20	20	10000	18	0.63	0.07	8.2E+07	3.2E+06	6.9E+07	24	4.8E+06	3.6E+07	10000	4.3E+07	10000	10000.0	16	0.5	1.7E+08	10000	7.7E+06
25	20	10000	12	0.65	0.05	8.8E+07	4.0E+06	8.9E+07	25	6.2E+06	5.3E+07	10000	4.2E+07	10000	10000.0	16	0.5	2.6E+08	10000	1.1E+07

C.14 Gas Sensor Array Drift

Table C.27: Summary of the results with Gas Sensor Array Drift ($\times 10^{13}, m = 13910, n = 128$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7.91186	0.12	0.08	0.68	0.57	14.06	18.21	0.03	0.0	0.01	0.04	0.03	0.01	0.17	0.21	0.13	0.04
3	5.02412	0.15	0.1	0.7	0.55	42.42	29.16	0.04	0.0	0.02	0.07	0.06	0.02	0.97	2.93	0.14	0.05
5	3.22394	0.22	3.39	1.27	0.5	21.81	33.61	0.07	0.0	8.15	0.39	0.13	0.03	8.45	4.99	0.11	0.03
10	1.65524	0.43	1.5	1.76	0.54	77.48	41.68	0.14	0.0	44.3	16.52	0.39	0.16	30.65	13.92	0.17	0.05
15	1.13801	-0.08	1.36	1.29	0.55	101.91	45.96	0.21	0.0	29.22	24.59	0.66	0.31	8.67	11.99	0.47	0.2
20	0.87916	1.46	1.01	1.62	0.34	135.38	51.73	0.28	0.03	46.2	9.82	1.07	0.43	13.95	5.77	0.6	0.23
25	0.72274	2.31	1.36	1.81	0.48	151.15	41.8	0.33	0.08	54.01	13.61	1.95	0.78	14.12	5.3	0.23	0.05
Mean:		**0.66**		**1.3**		**77.74**		**0.16**		**25.99**		**0.61**		**11.0**		**0.27**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7.91186	-0.0	0.0	0.07	0.02	10.17	45.46	5.83	0.41	-0.0	0.0	0.2	0.01	0.02	0.02	0.06	0.01
3	5.02412	-0.0	12.28	0.08	0.03	15.92	15.71	5.81	0.52	-0.0	0.0	0.28	0.04	0.05	16.16	0.07	0.02
5	3.22394	3.46	5.24	0.19	0.1	34.32	7.78	5.6	0.17	8.1	0.59	0.35	0.09	6.99	6.4	0.13	0.03
10	1.65524	4.02	10.5	0.37	0.15	74.31	14.14	6.96	0.28	32.31	13.99	0.81	0.24	6.44	8.46	0.29	0.13
15	1.13801	5.25	2.78	0.79	0.26	113.63	15.68	9.26	3.62	20.78	7.25	1.26	0.43	9.6	5.08	0.48	0.17
20	0.87916	6.25	5.75	1.13	0.48	148.13	21.46	14.68	3.73	21.78	8.56	2.18	0.46	14.85	7.76	0.67	0.25
25	0.72274	5.55	2.67	1.21	0.37	179.99	17.23	8.71	1.17	29.99	6.8	3.27	0.74	18.87	8.68	0.8	0.34
Mean:		**3.5**		**0.55**		**82.36**		**8.12**		**16.14**		**1.19**		**8.12**		**0.36**	

Table C.28: Clustering details with Gas Sensor Array Drift

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	30	9000	430	1.0	1.0	9.4E+07	1.1E+05	2.8E+05	20	3.7E+05	4.5E+05	9000	4.2E+07	9000	3000.0	16	0.5	8.1E+07	9000	2.7E+05
3	30	9000	350	0.87	1.13	9.7E+07	1.7E+05	6.2E+05	18	4.7E+05	6.3E+05	9000	4.2E+07	9000	3000.0	16	0.5	8.2E+07	9000	3.9E+05
5	30	9000	346	0.87	1.13	1.0E+08	2.8E+05	1.4E+06	22	9.7E+05	1.8E+06	9000	4.3E+07	9000	3000.0	16	0.5	8.3E+07	9000	9.7E+05
10	30	9000	210	1.4	0.6	1.2E+08	5.6E+05	4.1E+06	20	1.8E+06	3.3E+06	9000	4.6E+07	9000	3000.0	16	0.5	8.7E+07	9000	2.6E+06
15	30	9000	80	1.87	0.13	1.3E+08	8.3E+05	6.8E+06	23	3.1E+06	6.4E+06	9000	5.1E+07	9000	3000.0	16	0.5	9.0E+07	9000	4.0E+06
20	30	9000	43	0.2	1.8	1.0E+08	1.1E+06	1.1E+07	25	4.5E+06	1.0E+07	9000	5.1E+07	9000	3000.0	16	0.5	9.9E+07	9000	5.8E+06
25	30	9000	22	0.8	1.2	8.5E+07	1.4E+06	2.0E+07	27	6.1E+06	1.1E+07	9000	4.7E+07	9000	3000.0	16	0.5	1.1E+08	9000	7.5E+06

C.15 3D Road Network

Table C.29: Summary of the results with 3D Road Network ($\times 10^6, m = 434874, n = 3$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	49.13298	0.01	0.01	0.24	0.13	1.9	2.16	0.38	0.06	0.0	0.0	0.06	0.0	0.18	0.42	0.37	0.13
3	22.77818	0.01	0.01	0.28	0.13	13.59	27.4	0.54	0.01	0.0	74.5	0.1	0.01	0.8	0.85	0.41	0.11
5	8.82574	0.03	0.02	0.34	0.13	33.32	55.9	0.84	0.12	77.07	46.21	0.24	0.03	1.3	2.54	0.49	0.1
10	2.56661	0.19	0.22	0.43	0.18	111.62	127.83	2.11	0.4	52.14	53.8	1.65	0.26	2.02	2.47	0.78	0.25
15	1.27069	0.23	0.29	0.51	0.38	205.67	165.09	3.67	0.44	59.12	27.43	2.54	0.27	4.42	3.43	0.66	0.1
20	0.80865	0.59	0.59	0.66	0.42	173.25	112.73	5.2	0.6	45.69	23.7	4.64	0.67	5.17	3.14	0.78	0.11
25	0.59259	0.58	0.52	0.79	0.61	224.75	131.93	6.56	0.66	50.01	21.16	6.75	0.8	5.4	4.09	0.83	0.12
Mean:		**0.24**		**0.46**		**109.16**		**2.76**		**40.57**		**2.28**		**2.76**		**0.62**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	49.13298	0.0	0.0	0.11	0.03	11.76	11.65	19.41	1.36	0.0	0.0	0.33	0.05	0.03	0.09	0.01	0.0
3	22.77818	0.0	0.0	0.17	0.07	20.61	21.84	15.74	1.43	0.0	0.0	0.4	0.05	0.04	0.08	0.01	0.0
5	8.82574	0.0	0.0	0.51	0.1	32.07	24.5	14.88	0.98	0.0	0.0	0.83	0.06	0.12	0.17	0.02	0.0
10	2.56661	0.01	0.0	4.98	1.52	60.63	22.45	15.78	0.92	0.01	0.0	6.82	0.15	0.92	0.81	0.07	0.02
15	1.27069	0.0	0.0	4.94	2.47	123.99	34.74	16.93	1.24	0.0	1.88	9.35	0.69	2.5	3.25	0.1	0.02
20	0.80865	0.01	0.0	14.38	5.61	178.43	64.98	17.3	0.95	0.01	2.21	29.08	1.26	9.44	5.35	0.13	0.04
25	0.59259	0.07	1.03	12.62	5.03	275.78	73.71	16.03	0.94	1.62	1.57	31.19	1.34	19.7	9.55	0.15	0.07
Mean:		**0.01**		**5.39**		**100.47**		**16.58**		**0.23**		**11.14**		**4.68**		**0.07**	

Table C.30: Clustering details with 3D Road Network

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	40	100000	90	0.03	0.47	1.2E+08	3.5E+06	1.8E+07	27	5.4E+06	2.2E+07	10000	1.1E+08	10000	0.05	10	0.5	1.2E+08	10000	2.0E+06
3	40	100000	99	0.47	0.03	1.8E+08	5.2E+06	4.3E+07	29	8.7E+06	4.2E+07	10000	1.1E+08	10000	0.05	10	0.5	1.5E+08	10000	2.9E+06
5	40	100000	80	0.12	0.38	3.2E+08	8.7E+06	1.3E+08	32	1.6E+07	1.4E+08	10000	1.1E+08	10000	0.05	10	0.5	2.7E+08	10000	5.7E+06
10	40	100000	10	0.15	0.35	4.5E+08	2.1E+07	1.2E+09	35	3.5E+07	1.8E+09	10000	1.1E+08	10000	0.05	8	0.5	2.5E+09	10000	1.9E+07
15	40	100000	14	0.45	0.05	8.5E+08	3.4E+07	2.3E+09	34	5.0E+07	1.9E+09	10000	1.2E+08	10000	0.05	8	0.5	3.7E+09	10000	4.2E+07
20	40	100000	7	0.33	0.17	9.7E+08	4.8E+07	4.8E+09	35	7.0E+07	6.0E+09	10000	1.2E+08	10000	0.05	8	0.5	1.2E+10	10000	5.5E+07
25	40	100000	5	0.32	0.18	1.3E+09	5.8E+07	7.9E+09	32	8.0E+07	5.2E+09	10000	1.2E+08	10000	0.05	8	0.5	1.3E+10	10000	6.8E+07

C.16 Skin Segmentation

Table C.31: Summary of the results with Skin Segmentation ($\times 10^9, m = 245057, n = 3$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	1.32236	0.04	0.02	0.12	0.06	0.42	10.01	0.21	0.02	-0.0	0.0	0.01	0.0	0.02	0.1	0.08	0.04
3	0.89362	0.04	0.04	0.14	0.05	10.8	7.71	0.3	0.02	0.02	50.56	0.02	0.0	0.36	2.31	0.11	0.04
5	0.50203	0.09	0.29	0.15	0.07	16.38	14.76	0.47	0.06	14.34	20.25	0.03	0.0	1.86	5.17	0.11	0.04
10	0.25121	0.21	1.65	0.18	0.04	30.94	13.97	0.92	0.04	30.53	15.12	0.06	0.01	5.33	4.2	0.14	0.06
15	0.16964	0.98	1.83	0.17	0.05	40.88	18.18	1.4	0.23	30.04	18.17	0.1	0.01	6.03	3.24	0.18	0.06
20	0.12615	1.99	1.87	0.15	0.05	61.99	21.75	1.86	0.24	43.07	20.61	0.14	0.02	6.64	4.69	0.24	0.07
25	0.10228	3.15	1.8	0.18	0.05	64.36	21.86	2.28	0.35	28.76	16.08	0.18	0.02	6.79	3.27	0.19	0.06
Mean:		**0.93**		**0.16**		**32.25**		**1.06**		**20.97**		**0.08**		**3.86**		**0.15**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	1.32236	-0.0	0.0	0.06	0.01	90.1	31.19	3.7	0.21	-0.0	0.0	0.07	0.0	0.02	0.01	0.01	0.0
3	0.89362	-0.0	0.0	0.08	0.02	135.81	52.11	4.1	0.44	-0.0	0.0	0.17	0.02	0.04	0.03	0.01	0.0
5	0.50203	1.65	6.84	0.11	0.04	153.15	81.59	3.72	0.28	9.31	3.67	0.14	0.07	9.37	6.65	0.01	0.0
10	0.25121	8.98	5.69	0.2	0.08	77.69	36.59	4.94	0.13	16.1	2.94	0.3	0.05	17.64	7.56	0.02	0.0
15	0.16964	8.02	6.62	0.37	0.16	54.66	14.73	6.89	0.35	25.33	9.29	0.6	0.15	20.08	12.74	0.02	0.0
20	0.12615	9.74	3.31	0.59	0.19	78.9	16.03	5.95	0.42	27.45	8.19	0.68	0.23	29.2	10.48	0.02	0.01
25	0.10228	8.33	3.28	0.6	0.46	79.07	14.74	4.97	0.21	31.9	5.22	1.02	0.28	33.67	13.82	0.03	0.01
Mean:		**5.24**		**0.29**		**95.62**		**4.9**		**15.73**		**0.43**		**15.72**		**0.02**	

Table C.32: Clustering details with Skin Segmentation

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	30	8000	119	0.05	0.15	1.1E+07	2.0E+06	5.4E+06	30	4.9E+05	6.6E+06	8000	4.7E+07	8000	2.0	16	0.5	6.9E+07	8000	1.1E+06
3	30	8000	124	0.03	0.17	2.1E+07	2.9E+06	1.6E+07	47	1.1E+06	2.0E+07	8000	4.1E+07	8000	2.0	16	0.5	8.8E+07	8000	1.8E+06
5	30	8000	138	0.15	0.05	3.3E+07	4.9E+06	2.4E+07	38	1.5E+06	2.8E+07	8000	3.8E+07	8000	2.0	16	0.5	8.8E+07	8000	2.6E+06
10	30	8000	128	0.13	0.07	7.2E+07	9.8E+06	6.6E+07	48	3.9E+06	6.4E+07	8000	3.7E+07	8000	2.0	16	0.5	1.6E+08	8000	6.4E+06
15	30	8000	88	0.15	0.05	1.2E+08	1.5E+07	1.3E+08	49	5.9E+06	1.3E+08	8000	4.0E+07	8000	2.0	16	0.5	2.8E+08	8000	9.3E+06
20	30	8000	56	0.05	0.15	1.3E+08	2.0E+07	1.9E+08	60	9.5E+06	1.8E+08	8000	4.0E+07	8000	2.0	16	0.5	3.3E+08	8000	1.1E+07
25	30	8000	62	0.16	0.04	1.6E+08	2.5E+07	2.3E+08	52	1.0E+07	2.3E+08	8000	3.9E+07	8000	2.0	16	0.5	4.8E+08	8000	1.6E+07

C.17 KEGG Metabolic Relation Network (Directed)

Table C.33: Summary of the results with KEGG Metabolic Relation Network (Directed) ($\times 10^8, m = 53413, n = 20$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	11.3853	0.24	0.11	0.63	0.28	29.27	4.48	0.06	0.64	18.85	0.0	0.03	0.0	34.75	8.46	0.24	0.09
3	4.9006	0.56	0.26	0.5	0.23	174.75	19.91	0.09	0.0	124.79	0.0	0.08	0.0	141.74	7.79	0.3	0.09
5	1.88367	0.01	0.69	0.66	0.27	536.66	53.13	0.14	0.01	0.0	8.01	0.19	0.01	458.97	21.01	0.42	0.23
10	0.60513	0.04	1.83	0.68	0.19	1721.11	424.91	0.34	0.03	36.81	7.78	0.61	0.05	1508.43	54.07	0.51	0.14
15	0.35393	-0.03	4.41	0.9	0.2	2819.02	156.69	0.48	0.03	96.64	1.32	1.99	0.18	2556.76	106.8	0.39	0.1
20	0.25027	0.09	0.59	1.04	0.25	3935.45	653.74	0.64	0.06	169.77	4.28	2.8	0.74	3619.14	186.14	0.58	0.12
25	0.19289	1.04	0.88	1.17	0.23	4955.81	382.92	0.8	0.08	227.98	5.76	5.54	0.83	4694.61	217.03	0.55	0.31
Mean:		**0.28**		**0.8**		**2024.58**		**0.36**		**96.41**		**1.61**		**1859.2**		**0.43**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	11.3853	-0.0	9.0	0.01	0.68	56.76	10.82	2.84	10.63	18.85	0.0	0.22	0.07	0.0	0.0	0.01	3.63
3	4.9006	-0.0	50.11	0.06	0.02	243.76	22.11	3.51	0.54	124.79	0.0	0.32	0.05	0.01	0.03	0.02	0.0
5	1.88367	0.0	16.48	0.13	0.04	791.95	68.61	3.62	0.88	0.0	0.0	0.47	0.1	0.1	0.07	0.04	0.01
10	0.60513	8.43	15.9	0.3	0.11	2320.71	164.16	5.18	0.33	36.81	0.0	0.98	0.09	36.61	6.87	0.07	0.02
15	0.35393	6.46	17.67	0.36	0.22	3975.36	147.37	7.45	0.41	96.64	0.69	2.33	0.23	91.8	21.36	0.09	0.05
20	0.25027	10.65	25.17	0.64	0.29	5735.89	47.7	7.96	0.46	160.92	0.0	4.07	0.31	139.79	38.94	0.12	0.04
25	0.19289	8.36	11.33	1.02	0.47	7484.65	320.47	7.51	0.6	221.85	1.86	5.78	0.38	180.45	57.4	0.17	0.05
Mean:		**4.84**		**0.36**		**2944.16**		**5.44**		**94.27**		**2.02**		**64.11**		**0.08**	

Table C.34: Clustering details with KEGG Metabolic Relation Network (Directed)

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	20	53350	614	0.77	0.23	2.2E+08	4.3E+05	2.0E+06	25	2.7E+06	3.2E+05	10000	2.2E+07	10000	3.0	16	0.5	1.0E+08	10000	2.7E+05
3	20	53350	355	0.97	0.03	2.7E+08	6.4E+05	5.4E+06	19	3.0E+06	2.7E+06	10000	1.8E+07	10000	3.0	16	0.5	1.1E+08	10000	6.9E+05
5	20	53350	282	0.33	0.67	3.1E+08	1.1E+06	1.5E+07	16	4.4E+06	7.3E+06	10000	1.6E+07	10000	3.0	16	0.5	1.2E+08	10000	1.7E+06
10	20	53350	110	0.87	0.13	3.5E+08	2.6E+06	5.3E+07	22	1.1E+07	2.1E+07	10000	2.0E+07	10000	3.0	16	0.5	1.6E+08	10000	4.3E+06
15	20	53350	62	0.6	0.4	3.2E+08	3.7E+06	1.7E+08	19	1.5E+07	2.8E+07	10000	2.5E+07	10000	3.0	16	0.5	2.8E+08	10000	7.0E+06
20	20	53350	14	0.1	0.9	2.5E+08	5.0E+06	2.5E+08	21	2.2E+07	4.9E+07	10000	2.6E+07	10000	3.0	16	0.5	4.3E+08	10000	9.7E+06
25	20	53350	12	0.03	0.97	2.9E+08	6.2E+06	4.8E+08	20	2.7E+07	7.3E+07	10000	2.3E+07	10000	3.0	16	0.5	5.8E+08	10000	1.5E+07

C.18 Shuttle Control

Table C.35: Summary of the results with Shuttle Control ($\times 10^8, m = 58000, n = 9$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	21.34329	0.0	2.33	0.89	0.54	51.16	1.14	0.06	0.0	51.11	0.03	0.03	0.01	51.82	0.67	0.21	0.06
3	10.85415	3.66	1.45	0.8	0.34	195.45	10.97	0.09	0.0	100.52	32.99	0.04	0.02	196.64	1.8	0.21	0.07
4	8.8691	8.54	7.05	0.85	0.44	260.04	37.5	0.11	0.02	143.41	56.3	0.04	0.02	261.76	0.86	0.23	0.1
5	7.24479	0.18	7.31	0.84	0.31	299.22	37.03	0.14	0.05	38.69	35.03	0.06	0.02	341.24	4.43	0.21	0.05
10	2.83216	0.74	25.95	0.75	0.38	720.46	109.45	0.31	0.09	135.63	34.19	0.16	0.1	998.49	16.18	0.23	0.07
15	1.53154	4.46	2.41	1.27	0.44	1029.99	336.19	0.45	0.12	225.71	28.86	0.28	0.11	1883.71	66.41	0.25	0.06
20	1.06012	-0.45	1.36	1.13	0.28	823.88	245.56	0.59	0.04	307.27	31.55	0.26	0.07	2641.15	139.66	0.33	0.12
25	0.77978	3.53	56.3	1.11	0.32	1089.67	615.73	0.76	0.17	392.57	28.66	0.5	0.19	3466.23	202.52	0.42	0.1
Mean:		**2.58**		**0.96**		**558.74**		**0.31**		**174.37**		**0.17**		**1230.13**		**0.26**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	21.34329	5.04	15.24	0.01	0.0	54.09	0.7	4.57	0.36	51.07	0.03	0.36	0.04	0.01	2.02	0.01	0.0
3	10.85415	0.0	43.14	0.01	0.01	197.47	1.24	5.95	0.33	100.56	37.99	0.4	0.05	84.8	44.12	0.01	0.0
4	8.8691	15.19	40.54	0.02	0.01	263.16	0.58	5.65	0.42	143.53	43.61	0.46	0.04	15.28	49.49	0.01	0.0
5	7.24479	13.63	5.74	0.02	0.01	343.78	0.87	5.72	0.25	175.82	67.7	0.42	0.04	29.49	68.11	0.01	0.0
10	2.83216	22.05	30.77	0.06	0.03	1025.5	1.53	7.33	0.37	135.05	33.81	0.49	0.06	76.95	43.0	0.02	0.01
15	1.53154	30.94	12.83	0.17	0.12	1974.2	2.06	9.8	0.88	216.54	10.79	0.63	0.08	60.86	69.01	0.02	0.01
20	1.06012	26.77	17.9	0.19	0.09	2891.43	2.85	9.43	0.2	308.13	41.72	0.66	0.15	54.95	89.27	0.04	0.01
25	0.77978	14.45	16.27	0.3	0.09	3962.48	2.09	8.06	0.48	385.48	21.99	1.22	0.29	38.41	105.71	0.04	0.02
Mean:		**16.01**		**0.1**		**1339.01**		**7.06**		**189.52**		**0.58**		**45.09**		**0.02**	

Table C.36: Clustering details with Shuttle Control

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	15	57950	1416	0.75	0.75	5.5E+08	4.6E+05	3.5E+06	21	2.4E+06	4.6E+05	10000	3.8E+07	10000	3.0	16	0.5	1.0E+08	10000	3.3E+05
3	15	57950	915	0.65	0.85	6.3E+08	7.0E+05	5.4E+06	19	3.3E+06	8.7E+05	10000	4.0E+07	10000	3.0	16	0.5	1.1E+08	10000	5.0E+05
4	15	57950	904	0.6	0.9	7.1E+08	9.3E+05	7.0E+06	20	4.6E+06	1.2E+06	10000	4.2E+07	10000	3.0	16	0.5	1.1E+08	10000	6.7E+05
5	15	57950	787	0.6	0.9	8.0E+08	1.2E+06	9.9E+06	19	5.5E+06	1.4E+06	10000	4.4E+07	10000	3.0	16	0.5	1.1E+08	10000	8.6E+05
10	15	57950	359	0.1	1.4	9.4E+08	2.5E+06	3.4E+07	17	9.9E+06	7.5E+06	10000	4.9E+07	10000	3.0	16	0.5	1.3E+08	10000	2.3E+06
15	15	57950	351	1.1	0.4	9.7E+08	3.9E+06	5.0E+07	17	1.5E+07	1.9E+07	10000	5.3E+07	10000	3.0	16	0.5	1.6E+08	10000	4.4E+06
20	15	57950	188	0.6	0.9	1.0E+09	5.1E+06	5.9E+07	22	2.5E+07	3.5E+07	10000	5.5E+07	10000	3.0	16	0.5	1.6E+08	10000	7.7E+06
25	15	57950	102	0.35	1.15	9.4E+08	6.5E+06	1.1E+08	23	3.3E+07	4.4E+07	10000	5.2E+07	10000	3.0	16	0.5	2.7E+08	10000	8.8E+06

C.19 Shuttle Control (normalized)

Table C.37: Summary of the results with Shuttle Control (normalized) ($\times 10^1, m = 58000, n = 9$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	104.41601	0.21	0.16	0.18	0.11	14.73	13.71	0.17	0.04	10.73	12.02	0.01	0.0	4.1	10.58	0.11	0.05
3	73.28769	0.56	0.37	0.31	0.13	1.4	13.41	0.19	0.08	8.35	13.38	0.01	0.0	0.42	0.88	0.07	0.05
4	50.076	0.57	0.39	0.25	0.1	0.02	6.6	0.27	0.08	0.54	22.36	0.01	0.0	6.76	13.43	0.12	0.05
5	39.78043	1.47	0.79	0.32	0.1	0.86	9.7	0.32	0.07	12.94	26.62	0.01	0.0	1.67	1.58	0.11	0.04
10	15.04997	0.79	1.08	0.31	0.11	34.16	22.8	0.71	0.14	63.36	28.42	0.02	0.0	4.37	2.78	0.08	0.05
15	9.81804	2.67	1.97	0.21	0.11	88.68	38.77	1.28	0.24	32.71	40.53	0.03	0.0	7.63	3.03	0.08	0.04
20	7.233	1.64	2.0	0.28	0.11	21.68	56.63	1.63	0.22	38.97	48.08	0.04	0.0	6.63	4.49	0.08	0.04
25	5.86461	5.26	2.03	0.22	0.12	31.02	65.59	2.15	0.32	53.82	61.3	0.04	0.01	7.71	2.55	0.08	0.05
Mean:		**1.65**		**0.26**		**24.07**		**0.84**		**27.68**		**0.02**		**4.91**		**0.09**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	104.41601	0.0	15.41	0.01	0.01	0.99	1.96	0.43	0.12	0.0	15.13	0.02	0.0	0.07	13.39	0.0	0.0
3	73.28769	1.76	11.09	0.03	0.01	32.04	2.43	0.54	0.01	1.76	13.06	0.03	0.0	2.09	14.72	0.0	0.0
4	50.076	12.41	17.03	0.03	0.01	36.92	14.09	0.62	0.03	0.0	12.01	0.03	0.0	30.69	29.04	0.0	0.0
5	39.78043	2.72	9.08	0.04	0.02	33.66	13.6	0.71	0.04	0.83	2.45	0.05	0.01	25.43	41.88	0.01	0.0
10	15.04997	6.45	16.3	0.08	0.03	46.5	22.29	0.81	0.04	2.87	23.99	0.05	0.03	51.65	58.52	0.01	0.0
15	9.81804	10.69	7.06	0.13	0.05	53.8	63.43	0.64	0.04	7.82	17.34	0.08	0.04	44.92	40.22	0.01	0.0
20	7.233	13.48	5.32	0.19	0.06	117.02	190.87	0.55	0.03	7.97	24.7	0.16	0.09	58.76	33.04	0.01	0.0
25	5.86461	10.76	5.15	0.18	0.06	386.47	313.37	0.48	0.01	10.63	5.87	0.13	0.06	48.45	31.99	0.01	0.0
Mean:		**7.28**		**0.09**		**88.43**		**0.6**		**3.98**		**0.07**		**32.76**		**0.01**	

Table C.38: Clustering details with Shuttle Control (normalized)

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	20	2000	872	0.04	0.36	1.7E+07	7.1E+05	1.4E+06	37	1.5E+05	5.8E+05	2000	4.1E+06	2000	0.0003	2	0.5	4.8E+06	2000	2.5E+05
3	20	2000	1278	0.28	0.12	4.0E+07	1.3E+06	2.3E+06	27	1.6E+05	2.3E+06	2000	4.3E+06	2000	0.0003	2	0.5	5.6E+06	2000	3.3E+05
4	20	2000	974	0.24	0.16	4.9E+07	1.7E+06	3.1E+06	42	3.3E+05	3.1E+06	2000	4.6E+06	2000	0.0003	2	0.5	6.6E+06	2000	4.3E+05
5	20	2000	1144	0.19	0.21	7.0E+07	2.3E+06	4.8E+06	39	3.9E+05	4.5E+06	2000	4.8E+06	2000	0.0003	2	0.5	9.0E+06	2000	5.1E+05
10	20	2000	924	0.16	0.24	1.1E+08	5.5E+06	1.2E+07	28	5.6E+05	1.2E+07	2000	5.3E+06	2000	0.0003	2	0.5	9.9E+06	2000	9.8E+05
15	20	2000	370	0.32	0.08	1.5E+08	9.8E+06	2.0E+07	21	6.3E+05	1.7E+07	2000	4.9E+06	2000	0.0003	2	0.5	1.8E+07	2000	1.6E+06
20	20	2000	343	0.25	0.15	1.8E+08	1.3E+07	2.6E+07	19	7.6E+05	3.2E+07	2000	4.8E+06	2000	0.0003	2	0.5	3.0E+07	2000	2.1E+06
25	20	2000	229	0.08	0.32	2.0E+08	1.7E+07	3.3E+07	19	9.5E+05	3.1E+07	2000	4.5E+06	2000	0.0003	2	0.5	2.9E+07	2000	2.6E+06

C.20 EEG Eye State

Table C.39: Summary of the results with EEG Eye State ($\times 10^8, m = 14980, n = 14$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7845.09934	4.25	0.0	0.89	0.45	97.25	33.08	0.02	0.0	-0.0	0.93	0.0	0.0	98.06	0.24	0.08	0.02
3	1833.88058	0.0	65.98	0.6	0.46	327.74	86.37	0.02	0.0	227.91	81.38	0.01	0.0	747.5	1.08	0.06	0.02
4	2.23605	0.0	0.0	0.82	0.41	350642.24	88855.37	0.03	0.0	268809.8	128214.1	0.02	0.01	694466.19	499.31	0.05	0.01
5	1.33858	-0.0	6.52	0.93	0.35	585674.55	186919.07	0.04	0.0	449091.75	214188.85	0.03	0.01	1159957.25	4993.87	0.06	0.02
10	0.4531	0.0	0.01	0.9	0.38	1119056.99	644551.55	0.09	0.0	663435.01	663330.92	0.1	0.02	3386305.55	61885.8	0.08	0.02
15	0.34653	0.07	0.15	0.92	0.36	2261071.4	775757.28	0.11	0.01	1734685.37	862995.0	0.15	0.04	4344184.11	97469.29	0.13	0.02
20	0.28986	0.05	0.23	1.42	0.43	630877.35	1068598.65	0.14	0.02	2073832.6	1015966.35	0.26	0.09	5305346.5	194325.45	0.11	0.02
25	0.25989	0.12	0.07	1.06	0.34	702866.81	1017166.36	0.21	0.03	2312984.45	1150694.93	0.41	0.09	5857365.28	167835.2	0.13	0.03
Mean:		**0.56**		**0.94**		**706326.79**		**0.08**		**937883.36**		**0.12**		**2593558.8**		**0.09**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	7845.09934	17.49	20.02	0.0	0.0	98.37	0.02	1.92	0.56	-0.0	0.0	0.83	0.02	4.25	6.59	0.0	0.0
3	1833.88058	0.0	122.29	0.0	0.0	748.33	0.04	2.26	0.46	227.91	0.0	0.83	0.02	123.03	126.44	0.0	0.0
4	2.23605	0.0	0.0	0.01	0.0	695638.7	37.25	2.34	0.29	268809.8	0.0	0.84	0.02	0.02	63442.6	0.0	0.0
5	1.33858	14.95	14.95	0.01	0.01	1162076.58	71.7	2.66	0.25	449091.75	0.0	0.89	0.06	30.0	12.0	0.01	0.0
10	0.4531	-0.01	0.32	0.07	0.03	3432933.34	115.55	3.94	0.1	1326870.19	289134.66	0.93	0.03	190.17	74.52	0.03	0.01
15	0.34653	0.88	0.29	0.15	0.09	4488641.07	43.11	5.53	0.16	1734686.09	0.29	1.0	0.04	2.34	123.48	0.06	0.03
20	0.28986	0.78	0.54	0.21	0.12	5366270.87	120.44	4.82	0.34	2073833.78	0.66	1.16	0.1	304.43	147.96	0.08	0.03
25	0.25989	0.3	0.66	0.28	0.09	5985037.33	175.09	4.02	0.19	2312984.96	0.83	1.19	0.12	340.37	146.72	0.07	0.04
Mean:		**4.3**		**0.09**		**2641430.58**		**3.44**		**1020813.06**		**0.96**		**124.33**		**0.03**	

Table C.40: Clustering details with EEG Eye State

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	20	14979	4504	0.7	0.8	4.8E+08	1.2E+05	1.8E+05	20	6.1E+05	9.0E+04	8000	9.3E+06	8000	25.0	16	0.5	6.4E+07	8000	1.1E+05
3	20	14979	2504	0.05	1.45	5.7E+08	1.8E+05	4.0E+05	16	7.2E+05	1.3E+05	8000	8.8E+06	8000	25.0	16	0.5	6.5E+07	8000	1.5E+05
4	20	14979	2835	0.3	1.2	6.5E+08	2.4E+05	1.6E+06	21	1.3E+06	1.8E+05	8000	9.8E+06	8000	25.0	16	0.5	6.6E+07	8000	1.9E+05
5	20	14979	2783	0.85	0.65	6.9E+08	3.0E+05	3.4E+06	17	1.3E+06	3.4E+05	8000	1.1E+07	8000	25.0	16	0.5	6.8E+07	8000	7.4E+05
10	20	14979	1413	0.85	0.65	7.9E+08	6.0E+05	9.9E+06	20	2.9E+06	6.9E+06	8000	1.7E+07	8000	25.0	16	0.5	7.8E+07	8000	2.9E+06
15	20	14979	970	0.35	1.15	8.5E+08	9.0E+05	1.9E+07	21	4.7E+06	1.5E+07	8000	2.3E+07	8000	25.0	16	0.5	8.6E+07	8000	6.4E+06
20	20	14979	1120	1.4	0.1	8.7E+08	1.2E+06	3.4E+07	20	6.1E+06	2.4E+07	8000	2.0E+07	8000	25.0	16	0.5	1.1E+08	8000	9.0E+06
25	20	14979	454	0.2	1.3	7.9E+08	1.5E+06	4.8E+07	18	6.6E+06	3.2E+07	8000	1.7E+07	8000	25.0	16	0.5	1.1E+08	8000	8.1E+06

C.21 EEG Eye State (normalized)

Table C.41: Summary of the results with EEG Eye State (normalized) ($\times 10^1, m = 14980, n = 14$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	1.15267	0.0	3.14	0.56	0.27	25.4	0.01	0.13	0.06	25.4	0.01	0.01	0.0	27.5	1.26	0.06	0.02
3	0.82423	0.0	6.44	0.61	0.22	69.04	0.03	0.15	0.03	69.04	0.03	0.02	0.0	74.74	2.38	0.06	0.02
4	0.5429	0.0	6.11	0.72	0.31	152.48	0.06	0.22	0.09	152.47	0.06	0.02	0.01	160.89	4.6	0.07	0.02
5	0.28952	0.0	18.53	0.64	0.24	367.07	22.96	0.39	0.09	367.1	0.08	0.04	0.01	382.53	8.64	0.08	0.03
10	0.10269	-0.01	0.0	0.72	0.25	879.03	161.91	1.29	0.32	879.79	175.98	0.11	0.03	1188.62	3.96	0.09	0.03
15	0.07469	0.04	0.06	0.66	0.19	853.05	313.05	1.7	0.36	852.99	263.04	0.15	0.05	1639.36	26.21	0.11	0.03
20	0.06125	0.07	0.15	0.77	0.17	1044.53	390.42	2.68	0.61	1044.25	417.09	0.24	0.1	2007.07	52.77	0.11	0.03
25	0.05385	-0.2	0.11	0.97	0.13	1191.01	254.04	3.55	0.51	1187.56	339.48	0.3	0.07	2276.63	68.05	0.15	0.05
Mean:		**-0.01**		**0.71**		**572.7**		**1.26**		**572.33**		**0.11**		**969.67**		**0.09**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	1.15267	11.4	10.29	0.0	0.01	25.64	1.19	1.99	0.1	25.4	0.0	0.61	0.09	6.1	4.82	0.0	0.0
3	0.82423	27.59	22.03	0.01	0.01	73.56	1.84	2.11	0.07	69.04	0.04	0.66	0.09	38.62	6.92	0.01	0.0
4	0.5429	37.84	27.91	0.02	0.01	159.64	1.72	2.36	0.07	152.35	0.05	0.6	0.08	90.57	14.93	0.01	0.0
5	0.28952	105.16	54.36	0.03	0.01	381.15	2.61	2.75	0.27	367.07	0.11	0.67	0.2	261.69	37.78	0.02	0.01
10	0.10269	0.7	0.56	0.13	0.13	1218.5	8.26	4.19	0.19	879.68	79.92	0.75	0.09	568.99	170.56	0.04	0.01
15	0.07469	0.15	0.23	0.18	0.08	1685.07	13.69	6.05	0.2	853.24	170.74	0.76	0.1	413.56	258.62	0.08	0.03
20	0.06125	0.5	1.26	0.22	0.13	2062.84	12.35	4.71	0.16	1044.43	1.67	0.9	0.16	502.54	291.59	0.07	0.02
25	0.05385	0.7	0.61	0.29	0.1	2344.18	10.51	4.19	0.12	1191.05	110.44	0.97	0.14	565.67	344.68	0.09	0.03
Mean:		**23.01**		**0.11**		**993.82**		**3.54**		**572.78**		**0.74**		**305.97**		**0.04**	

Table C.42: Clustering details with EEG Eye State (normalized)

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	30	14979	2454	0.23	0.77	2.7E+08	6.9E+05	9.9E+05	18	5.5E+05	9.0E+04	8000	1.4E+07	8000	0.003	16	0.5	6.5E+07	8000	1.1E+05
3	30	14979	2034	0.53	0.47	3.2E+08	9.4E+05	1.5E+06	18	7.9E+05	5.2E+05	8000	1.3E+07	8000	0.003	16	0.5	6.6E+07	8000	6.3E+05
4	30	14979	2214	0.93	0.07	3.6E+08	1.5E+06	2.4E+06	20	1.2E+06	1.4E+06	8000	1.3E+07	8000	0.003	16	0.5	6.6E+07	8000	7.0E+05
5	30	14979	1580	0.73	0.27	3.8E+08	2.6E+06	4.0E+06	22	1.6E+06	2.5E+06	8000	1.4E+07	8000	0.003	16	0.5	6.9E+07	8000	1.3E+06
10	30	14979	928	0.5	0.5	4.4E+08	8.7E+06	1.4E+07	20	3.0E+06	1.0E+07	8000	1.8E+07	8000	0.003	16	0.5	7.8E+07	8000	4.3E+06
15	30	14979	518	0.33	0.67	4.6E+08	1.2E+07	2.0E+07	20	4.5E+06	1.7E+07	8000	2.3E+07	8000	0.003	16	0.5	8.7E+07	8000	8.2E+06
20	30	14979	451	0.73	0.27	4.6E+08	1.9E+07	3.3E+07	18	5.4E+06	2.2E+07	8000	2.1E+07	8000	0.003	16	0.5	1.0E+08	8000	8.6E+06
25	30	14979	418	0.93	0.07	4.7E+08	2.3E+07	4.0E+07	19	7.1E+06	3.7E+07	8000	1.8E+07	8000	0.003	16	0.5	1.1E+08	8000	1.1E+07

C.22 Pla85900

Table C.43: Summary of the results with Pla85900 ($\times 10^{15}, m = 85900, n = 2$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.74908	0.01	0.01	0.89	0.45	1.84	2.65	0.12	0.02	7.14	3.16	0.01	0.0	1.49	1.36	0.08	0.03
3	2.28057	0.02	0.02	0.58	0.42	9.0	8.72	0.1	0.03	0.01	20.75	0.02	0.0	0.57	3.06	0.11	0.03
5	1.33972	0.04	0.03	1.06	0.4	9.35	8.03	0.16	0.02	7.63	18.79	0.03	0.01	2.02	1.32	0.11	0.04
10	0.68294	0.12	0.17	0.92	0.4	13.68	9.38	0.29	0.03	21.1	12.44	0.07	0.02	4.0	1.73	0.18	0.06
15	0.46029	0.24	0.19	1.03	0.42	17.58	9.01	0.45	0.08	16.73	8.27	0.11	0.03	5.32	1.54	0.17	0.04
20	0.34988	0.29	0.13	0.81	0.39	20.07	6.98	0.58	0.11	13.01	7.03	0.14	0.03	5.2	1.48	0.21	0.05
25	0.28259	0.56	0.43	1.04	0.46	18.78	6.48	0.72	0.12	15.33	7.15	0.18	0.04	5.34	1.59	0.23	0.06
Mean:		**0.18**		**0.9**		**12.9**		**0.35**		**11.56**		**0.08**		**3.42**		**0.16**	

k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.74908	0.0	0.71	0.02	0.01	30.27	7.7	163.84	6.87	0.0	0.7	0.06	0.01	0.05	0.72	0.01	0.0
3	2.28057	0.0	0.0	0.09	0.04	69.0	13.09	160.56	7.95	0.0	0.0	0.08	0.01	0.03	0.07	0.01	0.0
5	1.33972	0.81	0.94	0.07	0.04	65.37	28.85	161.63	10.27	0.0	1.1	0.08	0.03	0.87	1.29	0.01	0.01
10	0.68294	0.49	0.85	0.24	0.08	32.94	9.6	181.45	8.86	0.42	0.5	0.25	0.1	0.96	0.94	0.03	0.01
15	0.46029	0.51	0.68	0.47	0.27	30.44	10.35	185.32	13.63	0.48	0.66	0.38	0.2	1.32	1.0	0.04	0.02
20	0.34988	0.46	0.69	0.56	0.24	27.52	7.65	201.73	9.83	0.66	0.71	0.46	0.25	1.81	0.78	0.06	0.02
25	0.28259	0.86	0.54	0.62	0.29	27.47	5.6	183.76	16.33	0.79	0.52	0.72	0.27	2.13	0.83	0.07	0.04
Mean:		**0.45**		**0.3**		**40.43**		**176.9**		**0.34**		**0.29**		**1.02**		**0.03**	

Table C.44: Clustering details with Pla85900

k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	40	14000	2394	1.0	0.5	3.9E+08	6.9E+05	4.0E+06	29	8.1E+05	4.8E+06	14000	2.6E+08	14000	2000.0	3	0.5	2.0E+08	14000	1.0E+06
3	40	14000	1386	1.4	0.1	6.6E+08	1.0E+06	1.1E+07	32	1.3E+06	1.4E+07	14000	2.6E+08	14000	2000.0	3	0.5	2.1E+08	14000	2.2E+06
5	40	14000	2182	1.05	0.45	1.0E+09	1.7E+06	1.7E+07	34	2.4E+06	1.8E+07	14000	2.6E+08	14000	2000.0	3	0.5	2.1E+08	14000	2.9E+06
10	40	14000	1204	1.2	0.3	2.0E+09	3.4E+06	6.0E+07	39	5.5E+06	8.8E+07	14000	2.7E+08	14000	2000.0	3	0.5	2.8E+08	14000	9.2E+06
15	40	14000	770	0.6	0.9	2.3E+09	5.2E+06	1.0E+08	38	8.1E+06	1.7E+08	14000	2.8E+08	14000	2000.0	3	0.5	3.4E+08	14000	1.6E+07
20	40	14000	428	1.4	0.1	2.9E+09	6.9E+06	1.5E+08	40	1.1E+07	2.7E+08	14000	2.9E+08	14000	2000.0	3	0.5	4.1E+08	14000	2.5E+07
25	40	14000	393	1.15	0.35	3.2E+09	8.6E+06	2.0E+08	43	1.5E+07	3.1E+08	14000	2.9E+08	14000	2000.0	3	0.5	5.3E+08	14000	3.0E+07

C.23 D15112

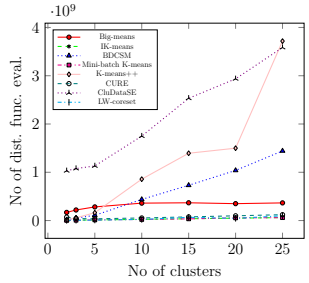
Table C.45: Summary of the results with D15112 ($\times 10^{11}, m = 15112, n = 2$)

k	f^*	Big-means				IK-means				BDCSM				Mini-batch K-means			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.68403	0.02	0.01	0.65	0.41	2.2	12.38	0.01	0.0	0.01	0.01	0.0	0.0	0.1	0.61	0.06	0.02
3	2.5324	0.04	0.02	1.03	0.43	8.51	7.61	0.03	0.0	0.03	0.03	0.0	0.0	0.25	4.11	0.06	0.02
5	1.32707	0.05	0.02	1.13	0.4	27.62	19.48	0.03	0.0	0.05	6.27	0.0	0.0	0.6	3.28	0.08	0.01
10	0.64491	0.1	0.21	1.08	0.37	25.11	14.92	0.11	0.01	0.77	3.3	0.02	0.01	4.61	3.1	0.11	0.04
15	0.43136	0.24	0.16	0.93	0.49	20.27	10.23	0.08	0.0	4.44	2.8	0.02	0.01	5.63	2.64	0.15	0.03
20	0.32177	0.55	0.33	1.25	0.41	18.5	4.75	0.2	0.04	1.97	1.0	0.02	0.01	6.31	1.78	0.16	0.03
25	0.25308	0.35	0.36	0.94	0.33	22.65	4.86	0.14	0.0	3.86	3.1	0.02	0.01	6.83	2.69	0.12	0.03
Mean:		**0.19**		**1.0**		**17.84**		**0.09**		**1.59**		**0.01**		**3.48**		**0.11**	

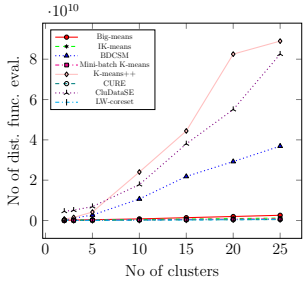
k	f^*	K-means++				CURE				CluDataSE				LW-coreset			
		ε		t		ε		t		ε		t		ε		t	
		med	std	med	std	med	std	med	std	med	std	med	std	med	std	med	std
2	3.68403	0.0	0.0	0.0	0.0	2.45	20.97	10.62	0.72	0.0	0.0	0.03	0.0	0.03	0.02	0.0	0.0
3	2.5324	0.0	0.0	0.01	0.0	30.34	15.04	10.42	0.71	0.0	0.0	0.06	0.0	0.07	0.04	0.01	0.0
5	1.32707	-0.0	0.0	0.01	0.01	52.77	26.89	11.11	0.65	-0.0	0.0	0.04	0.0	0.09	0.07	0.01	0.0
10	0.64491	0.63	1.38	0.03	0.01	40.22	13.27	12.59	0.63	3.83	1.46	0.08	0.01	1.7	2.67	0.01	0.01
15	0.43136	0.59	1.47	0.04	0.03	32.88	7.82	15.28	0.66	6.89	2.4	0.11	0.04	4.14	2.34	0.02	0.0
20	0.32177	1.99	1.56	0.07	0.02	36.48	8.62	13.69	0.74	1.37	3.24	0.09	0.01	3.77	1.94	0.02	0.01
25	0.25308	2.01	1.79	0.08	0.05	36.09	8.91	12.01	0.66	7.88	2.51	0.12	0.03	5.67	2.64	0.02	0.01
Mean:		**0.74**		**0.03**		**33.03**		**12.25**		**2.85**		**0.08**		**2.21**		**0.01**	

Table C.46: Clustering details with D15112

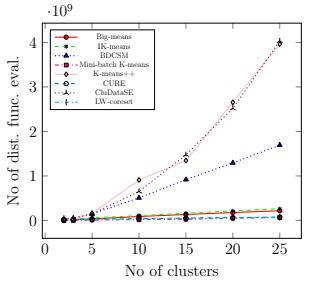
k	n_{exec}	Big-means					IK-means	BDCSM	Mini-batch K-means		K-means++	CURE		CluDataSE					LW-coreset	
		s	n_s	T_1	T_2	n_d	n_d	n_d	n_s	n_d	n_d	s	n_d	s	eps	min_pts	sf	n_d	s	n_d
2	15	8000	6558	0.95	0.55	7.6E+08	1.2E+05	2.5E+05	25	4.0E+05	4.5E+05	8000	7.6E+07	8000	200.0	16	0.5	6.4E+07	8000	2.7E+05
3	15	8000	8391	0.7	0.8	1.2E+09	1.8E+05	8.4E+05	25	6.0E+05	1.9E+06	8000	7.4E+07	8000	200.0	16	0.5	6.6E+07	8000	9.2E+05
5	15	8000	7163	0.7	0.8	1.4E+09	3.0E+05	1.1E+06	25	1.0E+06	1.4E+06	8000	7.6E+07	8000	200.0	16	0.5	6.7E+07	8000	1.1E+06
10	15	8000	3244	0.9	0.6	2.2E+09	6.0E+05	3.0E+06	35	2.8E+06	8.3E+06	8000	8.0E+07	8000	200.0	16	0.5	7.3E+07	8000	3.1E+06
15	15	8000	1584	1.35	0.15	2.5E+09	9.1E+05	6.3E+06	34	4.1E+06	1.2E+07	8000	8.4E+07	8000	200.0	16	0.5	8.2E+07	8000	5.7E+06
20	15	8000	1371	1.35	0.15	2.8E+09	1.2E+06	1.1E+07	33	5.3E+06	1.8E+07	8000	8.2E+07	8000	200.0	16	0.5	9.5E+07	8000	7.1E+06
25	15	8000	946	0.85	0.65	3.1E+09	1.5E+06	1.2E+07	26	5.2E+06	2.4E+07	8000	8.0E+07	8000	200.0	16	0.5	1.1E+08	8000	9.2E+06



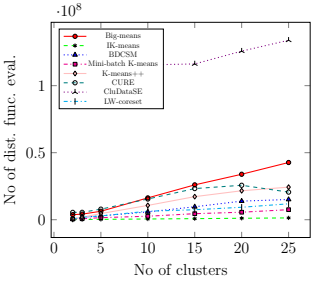
(a) CORD-19 Embeddings



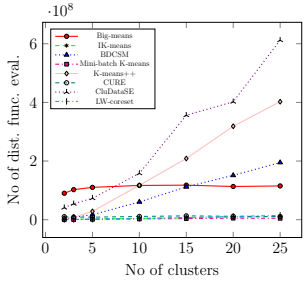
(b) HEPMASS



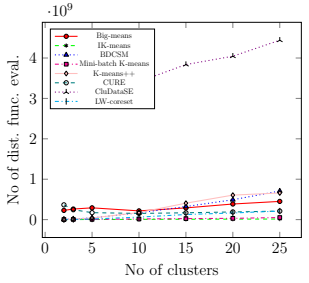
(c) US Census Data 1990



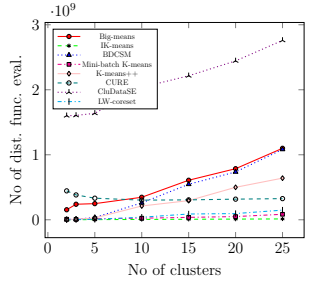
(d) Gisette



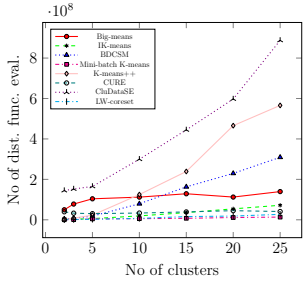
(e) Music Analysis



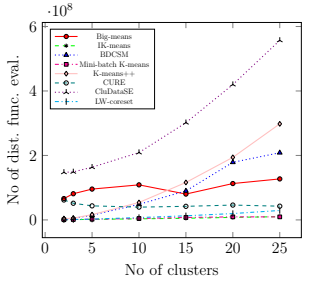
(f) Protein Homology



(g) MiniBooNE Particle Identification

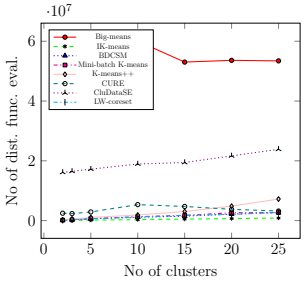


(h) MiniBooNE Particle Identification (normalized)

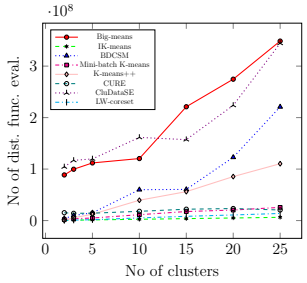


(i) MFCCs for Speech Emotion Recognition

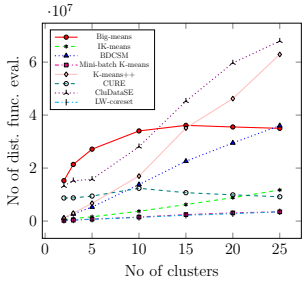
Figure C.1: Distance function evaluations. Set 1



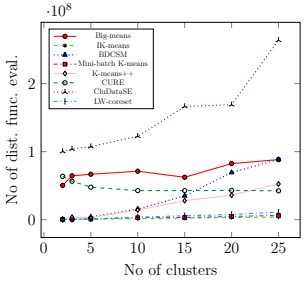
(a) ISOLET



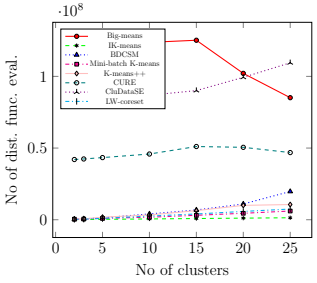
(b) Sensorless Drive Diagnosis



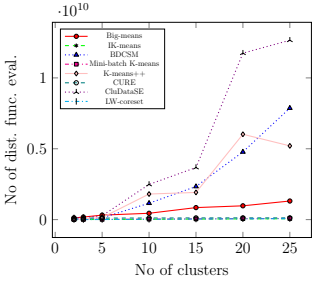
(c) Sensorless Drive Diagnosis (normalized)



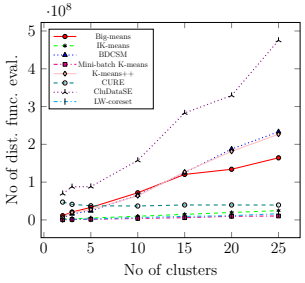
(d) Online News Popularity



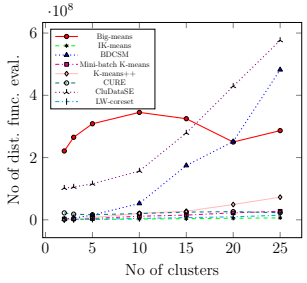
(e) Gas Sensor Array Drift



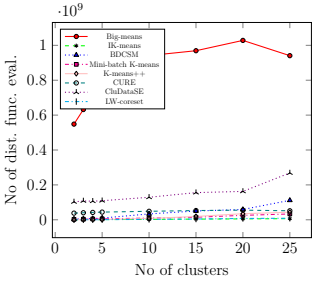
(f) 3D Road Network



(g) Skin Segmentation

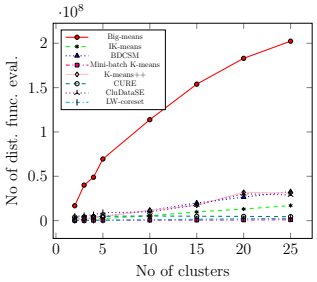


(h) KEGG Metabolic Relation Network (Directed)

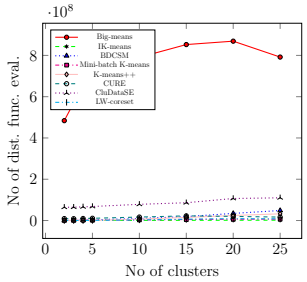


(i) Shuttle Control

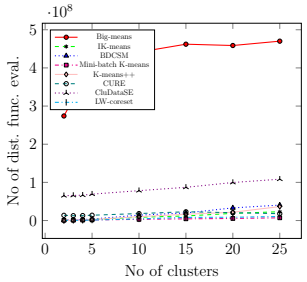
Figure C.2: Distance function evaluations. Set 2



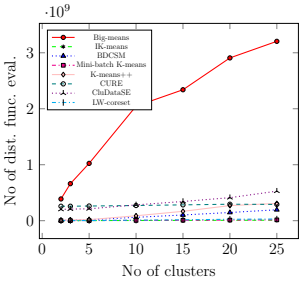
(a) Shuttle Control (normalized)



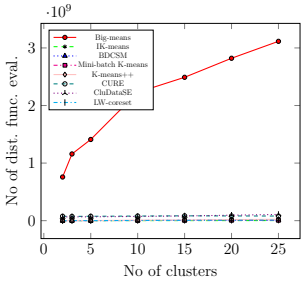
(b) EEG Eye State



(c) EEG Eye State (normalized)



(d) Pla85900



(e) D15112

Figure C.3: Distance function evaluations. Set 3

VITA

Ravil Mussabayev was born in Almaty, Kazakhstan in 1997 and developed a keen interest in math, physics, and computer programming from a young age. He excelled in physics, winning various Olympiads at district, city, and national levels. After completing high school in 2014, he pursued Engineering in Mathematical and Computer Modeling at Kazakh-British Technical University in Almaty, where he graduated with honors in 2018. His bachelor's thesis was on traffic light control using reinforcement learning and a novel clustering algorithm he created, which culminated in a publication.

During his undergraduate studies, Ravil also published research on controlling robotic arms with computer vision and contributed to text-to-speech technologies for people with disabilities. He engaged in creating mobile apps and freelancing as a software developer.

In 2018, he moved to the United States to start his PhD in Mathematics at the University of Washington in Seattle, under Professor Gunther Uhlmann, on a full scholarship. His research there focuses on developing effective and efficient scalable clustering algorithms for big data. Ravil has also taught undergraduate math courses and mentored students.

His considerable contribution to science includes the publication "How to Use K-means for Big Data Clustering?". In this work, a novel state-of-the-art big data clustering method was proposed that significantly excelled existing ones. This research was recognized in the journal Pattern Recognition. Ravil has also given 5 talks at international conferences.

Ravil is interested in the fields of data science, global optimization, machine learning, big data clustering, and natural language processing. Currently in the final year of his PhD, he aims to graduate in 2024 and pursue a research career, applying his expertise in machine learning and data science to address real-world challenges.