On Parallelization of Categorical Data Clustering

Bahareh Badiei

A Thesis
in
The Department of
Computer Science and Software Engineering
Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

May 2023

CONCORDIA UNIVERSITY

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Bahareh Badiei

Entitled: On Parallelization of Categorical Data Clustering

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (M. Comp. Sc.)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________ Chair
Dr. Sudhir Mudur

______________________ Examiner
Dr. Tristan Glatard

______________________ Examiner
Dr. Sudhir Mudur

______________________ Supervisor
Dr. Dhrubajyoti Goswami

______________________ Supervisor
Dr. Nematollaah Shiri

Approved by ______________________
Dr. Lata Narayanan, Chair of Department

____
Dr. Mourad Debbabi, Dean of Faculty of Engineering and Computer Science

Abstract

On Parallelization of Categorical Data Clustering

Bahareh Badiei

We study parallelization of categorical data clustering algorithms in an MPI platform. Clustering such data has been a daunting task even for sequential algorithms, mainly due to the challenges in finding suitable similarity/distance measures. We propose a parallel version of the k-modes algorithm, called PV3, which maintains the same clustering quality as produced by the sequential approach while achieving reasonable speed-ups. PV3 is programmed to ensure deterministic processing in a parallel environment. To produce better clustering results, we then develop an initialization method called Revised Density Method (RDM) based on the notion of density. Additionally, we develop variants of the RDM method to further enhance its performance. we then study effective ways to parallelize RDM and its variants. To further exploit parallelism opportunities, we develop an Ensemble Parallelizing Process (EPP) framework. This framework can be used with any desired initialization/clustering algorithms with different levels of parallelism. Using our different RDM initialization techniques along with the PV3 algorithm in the EPP framework, we then build an RDM realization of EPP, called RDM EPP. The result of our numerous experiments using benchmark categorical datasets indicate the quality metric of RDM EPP to be among the top three sequential k-modes based clustering algorithms. In terms of speed up, the results indicate to be 7 times faster for some datasets, though much larger datasets are required for a more comprehensive scalability study of RDM EPP.

Acknowledgements

I would like to express my gratitude to my esteemed professors, Dr. Dhrubajyoti Goswami and Dr. Nematollaah Shiri, for their unwavering support throughout my academic journey, particularly during my pregnancy and postpartum period. Their understanding and flexibility have been instrumental in enabling me to continue with my studies without undue difficulty.

I also would like to extend my heartfelt appreciation to my beloved husband, Homam Hosseini, whose tireless support, encouragement, and commitment to my success have had an important role in making the completion of this thesis possible. Without his invaluable assistance and belief in my abilities, I would not have been able to achieve this milestone in my academic career.

Table of Contents

List of Figures vii

List of Tables viii

Chapter 1. Introduction 1

1.1. Clustering Categorical Data 1
1.2. Thesis Motivation 1
1.3. Thesis Contributions 2
1.4. Thesis Outline 3

Chapter 2. Background, Concepts, and Definitions 4

2.1. Clustering Algorithms 4
2.2. Review of the K-Modes Algorithm 5
2.3. Clustering Validation Metrics 7
2.4. Parallel Computing 8
2.6. Summary 10

Chapter 3. Related Work 12

3.1. Cao's Algorithm 12
3.2. Khan and Ahmad's Algorithms 13
3.4. K-PbC Algorithm 16
3.5. Other K-Modes Based Clustering Algorithms 17
3.6. Parallel K-Means Using MPI 19
3.7. Parallel K-Means Based on Spark 21
3.8. Parallel K-Modes Based on MapReduce 21
3.9. A Cluster Ensemble Framework for Categorical Data 22

Chapter 4. Proposed Solution Techniques 23

4.1. Parallel Algorithm Version 1 (PV1) 23
4.2. Parallel Algorithm Version 2 (PV2) 26
4.3. Parallel Algorithm Version 3 (PV3) 28
4.4. RDM Algorithm and Variants 31

4.4.1. RDM Initialization Method 32
4.4.2. GLMD RDM Initialization 34
4.4.3. SCAV RDM Initialization 36
4.4.4. PRM RDM Initialization 37
4.4.5. SIG RDM Initialization 38
4.4.6. SIG-SCAV RDM Initialization 38
4.5. Parallel RDM Initialization and Parallel V3R (PV3R) 39
4.6. Parallel SIG RDM Initialization 41
4.7. Proposed Solution Framework (EPP) 43
4.8. Summary 45
Chapter 5. Experiments and Results 46
5.1. Datasets for Experimental Evaluations 46
5.2. Parallel K-Modes (PV1, PV2, PV3, and PV3R) 46
5.2.1. Speed-up 47
5.3. RDM 51
5.3.1. RDM vs. Cao 52
5.3.2. RDM, its Variants and RDM Ensemble 54
5.3.3. RDM Ensemble vs. Other Algorithms 56
5.3.4. RDM Realization of EPP (RDM EPP) 61
5.4. Summary 64
Chapter 6. Conclusion and Future Works 66
Reference 68
Appendix 72
Appendix A: Accuracy, Precision, and Recall of Compared Algorithms 72

List of Figures

Figure 3.1 Three MPI Processes Holding the Categorical data objects of Cluster $C1$ 20

Figure 4.1 PV1's Schema for k = 2 and p = 2 23

Figure 4.2 PV2's Schema for k = 2 and p = 3 26

Figure 4.3 PV3's Schema for k = 2 and p = 3 28

Figure 4.4 EPP Schematic.. 43

Figure 4.5 RDM Realization of EPP (RDM EPP).. 44

Figure 5.1 Maximum Speed-up Comparison Between PV1, PV2, PV3, and PV3R.................. 48

Figure 5.2 PV2's Speed-up... 49

Figure 5.3 PV3's Speed-up... 50

Figure 5.4 PV3R's Speed-up 51

Figure 5.5 Accuracy of RDM vs. Cao 52

Figure 5.6 Precision of RDM vs. Cao.. 53

Figure 5.7 Recall of RDM vs. Cao 54

Figure 5.8 Accuracy of Compared Algorithms.. 57

Figure 5.9 Precision of Compared Algorithms 59

Figure 5.10 Recall of Compared Algorithms... 60

Figure 5.11 SIG RDM's Speed-up (a Close Approximation of RDM EPP's Speed-up)............. 64

List of Tables

Table 3.1 Cluster C1 16
Table 3.2 Cluster C2 16
Table 4.1 Categorical Dataset 29
Table 4.2 Frequency Table of C1 29
Table 4.3 Frequency Table of C2 29
Table 4.4 Density-Object_List in the First Iteration 33
Table 5.1 Experimental Datasets 46
Table 5.2 Speed-up 47
Table 5.3 PV2's Speed-up vs. #Processes 49
Table 5.4 PV3's Speed-up vs. #Processes 50
Table 5.5 PV3R's Speed-up vs. #Processes 50
Table 5.6 Accuracy of RDM vs. Cao 52
Table 5.7 Precision of RDM vs. Cao 53
Table 5.8 Recall of RDM vs. Cao 54
Table 5.9 Accuracy of RDM's Variants and RDM Ensemble 55
Table 5.10 Precision of RDM's Variants and RDM Ensemble 55
Table 5.11 Recall of RDM's Variants and RDM Ensemble 56
Table 5.12 Accuracy of Compared Algorithms 57
Table 5.13 Precision of Compared Algorithms 58
Table 5.14 Recall of Compared Algorithms 60
Table 5.15 Lymphography - RDM EPP for Best p = 5 61
Table 5.16 Vote - RDM EPP for Best p = 8 62
Table 5.17 Breast cancer - RDM EPP for Best p = 8 62
Table 5.18 Car evaluation - RDM EPP for Best p = 9 62
Table 5.19 Chess - RDM EPP for Best p = 10 63
Table 5.20 Mushroom - RDM EPP for Best p = 10 63
Table 5.21 Nursery - RDM EPP for Best p = 10 63
Table A.1 Accuracy of Compared Algorithms 72
Table A.2 Precision of Compared Algorithms 73
Table A.3 Recall of Compared Algorithms 73

Chapter 1. Introduction

1.1. Clustering Categorical Data

Clustering is an unsupervised data mining technique which tries to put similar or homogenous (based on a similarity measure) unlabeled data objects into a group or cluster. The goal is to form clusters with as much intra-cluster similarity and inter-cluster dissimilarity as possible. By doing so, we can obtain meaningful knowledge about the data through identifying hidden patterns in it which can then be used in numerous application areas such as social network analysis [1], biological data analysis [2], dynamic trend detection [3], data summarization [4], and collaborative filtering [5], to name a few.

Categorical data is a type of data which can be divided into categories or groups. These categories are based on qualitative characteristics such as hair color, gender, and etc. Clustering categorical datasets is a challenging and subjective task due to its nature as there is no agreed universal distance/dissimilarity measure for such data (e.g., Euclidean distance is widely used for numerical data clustering). Looking at the data from different perspectives may change the way we cluster our data. A very simple example is clustering different cars based on their price, gas consumption, trim, or brands. In this case, different clustering results may be acceptable, and the choice depends on the user's application.

1.2. Thesis Motivation

Considering the vast amount of real-life data in today's applications, we need suitable tools and techniques to be able to extract valuable knowledge from it. This data could be numerical, categorical, or mixed numerical and categorical. Many clustering algorithms have been proposed to address this need. One of the most popular partitional clustering algorithms for categorical data is *k-modes* due to its linear complexity and simplicity. K-modes is an extension of the *k-means* algorithm. K-means is a method proposed to cluster numerical data while k-modes work with categorical data. In this research, we investigate parallelization of categorical data clustering, motivated by the repetitive nature of the k-modes algorithm which makes it a suitable candidate to be parallelized. On top of that, with the increasing amount of produced data, we wish to take advantage of available computing resources to speed-up the clustering process.

Similar to k-means, the resulting clustering quality in k-modes is affected by the selection of initial modes or centroids – a poor selection can yield poor results. Partitional clustering algorithms, in general, assume a given number of clusters [6] as input and select *k* random initial cluster centers called centroids. Using a dissimilarity measure, a k-modes algorithm forms clusters around these initial centroids and then tries to refine them in an iterative process to increase homogeneity inside the clusters and heterogeneity between clusters. The problem though is that in most cases, different initial centroids lead to different final results which are not the global optima but only local optima based on the initial centroids selected. In addition, producing different results in different runs make the results of k-modes less stable. To overcome this issue, numerous k-modes initialization algorithms have been proposed for traditional sequential clustering of categorical data, most notably k-PbC [7], Khan and Ahmad's [8], and Cao's [9] Algorithm. As part of this research, we adopt some of these initialization methods and develop parallelization algorithms for categorical data clustering.

1.3. Thesis Contributions

The main contributions of this thesis are as follows:

1. Exploring different ways of parallelizing the k-modes algorithm, and proposing the PV3 algorithm which is designed to produce the same clustering quality as produced by the sequential counterpart while achieving a reasonable speed-up in a parallel environment.

2. Proposing a new algorithm called Revised Density Method (RDM) to address the issue of the k-modes' *random* initialization which as discussed affects the clustering quality. We then extend RDM and develop different versions of it.

3. Parallelizing RDM, and plugging it into PV3 to introduce a new parallel algorithm, called PV3R, which is a parallel k-modes clustering algorithm with non-random initialization.

4. Proposing an Ensemble Parallelizing Process (EPP) framework which allows us to integrate multiple clustering solutions. This, in turn, improves clustering quality for categorical data. This framework has two levels of parallelism: one horizontally at the breadth level, and the other vertically at the depth.

5. Building an RDM realization of EPP framework (RDM EPP) which allows us to combine and use all our previously developed techniques including different RDM initialization methods, and the PV3 clustering algorithm. The quality of RDM EPP's clustering result (RDM ensemble) is better than all individual RDM variants, and stands among the top three best k-modes based clustering algorithms while enjoying easy implementation and parallelization.

6. Parallelizing two variants of RDM algorithm called SIG RDM and SIG-SCAV RDM which contributes significantly in achieving a reasonable overall speed-up for our RDM realization of EPP (RDM EPP).

7. Conducting numerous experiments to evaluate the quality of RDM, its variants, and RDM ensemble. We also evaluate the RDM EPP's speed-up. For these experiments, the standard benchmark datasets from UCI Machine Learning Repository [22] have been used.

1.4. Thesis Outline

The rest of this thesis report is organized as follows. Chapter 2 presents background knowledge, basic concepts, and techniques related to data clustering and parallel computing. Chapter 3 is a review of the related work. Chapter 4 elaborates on the implementation of various versions of parallel k-modes in MPI platform, a new initialization method for k-modes (RDM) and some of its variants, parallel RDM (PV3R), parallel SIG RDM, the EPP framework, and finally its RDM realization (RDM EPP) which receives inputs from the aforementioned k-modes clustering outputs. In chapter 5, we report the results of our numerous experiments carried to evaluate the performance of the individual components mentioned above as well as RDM EPP. Finally in chapter 6, we provide concluding remarks and discuss possible future research directions.

Chapter 2. Background, Concepts, and Definitions

2.1. Clustering Algorithms

There is a variety of proposed clustering algorithms. Each tries to tackle the clustering issue in its own unique way. In what follows, we begin with a classification of some of their approaches and discuss their pros and cons.

• Partitional Clustering Algorithms

The family of partitional clustering algorithms, also called centroid-based, usually receive a user-defined parameter, *k*, as the number of partitions or clusters. They then divide the dataset into *k* clusters, each initially assigned a centroid as its representative. This centroid can either be a virtual data object or a real object. In an iterative process, the goal is to move the objects between the clusters to a point where the total distance between the data objects and their corresponding centroids reaches a threshold, or no data object can change its membership anymore [41].

The main advantages of partitional clustering algorithms are [8]:
1. Fast (linear complexity in the number of data objects).
2. Fairly scalable.
3. Easy to implement.
4. Guaranteed convergence.

Major drawbacks of partitional clustering algorithms are [41]:
1. High sensitivity to the centroids assigned in the initialization phase and sensitivity to outliers.
2. The need to specify the number *k* of clusters, as there may be no prior knowledge of the correct value.

k-means and k-modes are well-known examples of such algorithms. The k-modes algorithm was employed to analyze the clustering of categorical data in this report.

• Density-Based Clustering Algorithms

These algorithms divide the dataset space into high density regions (clusters) separated by low density ones. A basic example of this kind of clustering is the DBSCAN algorithm which has two input parameters: *minPts* and *eps* (ε). The first parameter, minPts is the minimum number of data objects in a region which is required to be considered as a cluster, and the second parameter is a distance measure used for locating data objects in a neighborhood or the maximum radius of the neighborhoods [42].

There are two main concepts in this method: density reachability and density connectivity. Density reachability is when a data object is reachable from another, if it lies within the ε distance. Density connectivity, on the other hand, is when a data object is reachable via a sequence of other reachable data objects. The collection of such data objects forms a cluster [42].

Based on these definitions, there are three types of data objects considered in DBSCAN clustering: core, border, and noise. A core data object, *d*, is an object with at least *minPts* many data objects within the radius ε from *d*. A border data object is an object with at least one core object at distance ε, and a noise data object is neither a core nor a border [42].

DBSCAN uses these definitions in an iterative process to perform the clustering and produce the final results. Density-based algorithms can identify clusters with arbitrary shape and handle datasets with high dimensionality. In addition, they do not need repeated executions to produce the output. That said, their main drawback is requiring the user defined parameters (*minPts* and ε). While partitional algorithms also need a user defined parameter *k*, the required parameters in density-based algorithms are less intuitive and more difficult to provide [41].

- **Hierarchical Clustering Algorithms**

In these algorithms, clusters are formed in a hierarchical manner. Hierarchical clustering algorithms are further classified into two main categories: agglomerative and divisive. In the former, which is a bottom-up approach, we start by putting each data object in a separate cluster and then proceed to merge these clusters until we obtain a single one. On the other hand, in a divisive or a top-down approach, we start by one cluster which is the input dataset and then repeatedly divide it into smaller clusters [41].

The main advantages of this type of clustering algorithms are as follows: first, they need no user input. Second, they do not need to be executed multiple times because their results are stable. The main disadvantage though is the lack of scalability which makes them inefficient and expensive to use for clustering large datasets [41].

In the next section, we will review the k-modes algorithm which is a partitional clustering algorithm adapted to work with categorical data.

2.2. Review of the K-Modes Algorithm

K-modes is a partitional clustering algorithm specifically designed for clustering categorical data. It is, in fact, an adapted version of the k-means algorithm which differs in two key aspects: the dissimilarity measure employed (Euclidean distance in k-means and Hamming distance in k-modes), and the usage of modes instead of means within the k-modes algorithm. Modes simply refer to the most frequent attribute values for each attribute category.

- **Preliminaries**

a) Hamming Dissimilarity (Distance) Measure

This is a simple dissimilarity measure which counts the total number of mismatches between the attribute categories of two data objects [10], assigning equal importance or

weight to each category of attributes.
Let *X* and *Y* be two categorical data objects described by m categorical attributes:

$X = \{x_1, x_2, \dots, x_m\}$
$Y = \{y_1, y_2, \dots, y_m\}$

The distance between them is defined as follows:

$$d(X, Y) = \sum_{j=1}^{m} \delta(x_j, y_j) \qquad (1)$$

Where $\delta(x_j, y_j) = \begin{cases} 0, & (x_j = y_j) \\ 1, & (x_j \neq y_j) \end{cases}$

b) Cost Function in K-Modes

As mentioned earlier, in every partitional clustering algorithm, we need the number of clusters, *k*, as the input parameter to start with. K-modes is no exception. So, we have *k* clusters, and for each cluster, we define a centroid. Suppose, $Q = \{q_1, q_2, \dots, q_k\}$ is the set of modes. Using Equation 1, the cost function would be [10]:

$$E(Q) = \sum_{j=1}^{k} \sum_{i=1}^{n} d(o_i, q_j) \qquad (2)$$

Where o_i is the i^{th} data object in a total of *n* objects, and q_j is the corresponding cluster centroid to which o_i belongs. The aim in k-modes algorithm is to minimize the cost function in Equation 2.

• K-Modes Steps

K-modes was first introduced by Huang in 1997 [10]. The steps are as follows:

1. Select an initial cluster centroid for each of the *k* clusters.
2. Go through the data objects one at a time, and put them in a cluster that has the closest centroid to the data object in question. Update the cluster centroid after allocation of the data objects, and re-compute *k* new modes (centroids) for all the clusters.
3. Calculate the dissimilarity measure again, this time, between the new centroids and data objects, and allocate the data objects to the closest cluster based on their distance from the corresponding centroid. Update the cluster centroids.
4. Repeat step 3 until either the memberships of the data objects in clusters do not change, or we reach a certain threshold for the cost function in Equation 2.

2.3. Clustering Validation Metrics

There are three types of validation metrics for evaluating clustering quality: internal, external, and relative validation metrics. The first type measures the quality of clustering results based on the notation of compactness (intra-cluster distance) and separation (inter-cluster distance). The second type compares the results of the clustering to some priori knowledge indicated by experts or ground truth. It is used to choose the most suitable clustering algorithm for a given dataset [7]. The third type measures the quality of clustering results by using different parameter values for the same clustering algorithm [7].

In this thesis research, three types of external validation metrics were used to measure and compare the produced results' clustering quality to those of the others including: accuracy, precision, and recall.

- **Accuracy**

Accuracy measures how close the clustering result is to the ground truth. It is defined as the proportion of all the correctly clustered data objects to the total number of objects in a dataset [7].

$$AC = \frac{\sum_{i=1}^{k} TP_i}{n} \qquad (3)$$

In Equation 3, TP_i is the number of true positive (correctly classified) members of cluster *i*, and *n* is the total number of objects in a dataset.

- **Precision**

Precision measures how close the clustered data objects are to each other. It is defined as the proportion of all the correctly classified objects in a cluster to the total number of objects in the same cluster. To calculate the precision of a clustering result, we compute the average of the precisions of all the clusters [7].

$$PR = \frac{\sum_{i=1}^{k} (\frac{TP_i}{n_i})}{k} \qquad (4)$$

Where n_i is the number of data objects in cluster *i*.

- **Recall**

Recall is the proportion of correctly classified data objects in a cluster to the total number of the relevant data objects. In other words, recall indicates how successful the algorithm performed in finding all the relevant data objects of suitable clusters. To calculate recall, we compute the average of the recalls of all the clusters [7].

$$RE = \frac{\sum_{i=1}^{k} \left(\frac{TP_i}{TP_i+FN_i}\right)}{k} \qquad (5)$$

Where the false negative for cluster i or FN_i is the number of all the relevant data objects in cluster i that were incorrectly classified across all clusters other than cluster i.

2.4. Parallel Computing

This section will provide an overview of parallel computing, which is closely connected to and utilized within the context of this report.

• Distributed Memory Systems

Distributed memory architecture is one of the diverse parallel computing platforms available. In a distributed memory system, every process has its own local private memory which can be on the same physical machine and/or across any number of machines [11]. The communication between processes is explicit and hence, the programmer is responsible for determining the parallel tasks and exploiting parallelism using the subroutines in the libraries. Among various types of distributed memory systems, we have used the Beowulf cluster which is designed to provide cost-effective parallel computing capabilities by utilizing commodity hardware and open-source software.

• MPI Programming Model

Message Passing Interface (MPI) model is the most popular scheme in the distributed memory platform [18]. It is a standard and portable means of exchanging messages between the nodes in a distributed memory architecture which we have used in implementing our parallel algorithms.

Below are some of the advantages of the MPI model [37]:

1. MPI is known for its ability to keep its processes alive during the running of the system. In addition, there is no need to read the data several times from the disk. This quality makes MPI suitable for iterative jobs.
2. MPI does not need a server to control the data transfer between the contributors. This leads to accessibility of the resources by other contributors in the network [38].
3. MPI is scalable, i.e., it has the ability to add nodes dynamically without performance degradation.

Here are some main drawbacks of the MPI model:

1. MPI suffer from security issues and require high bandwidth usage [37].
2. Since the responsibility of coding in parallel is fully on the programmers, it can be a double-edged sword. Consequently, one can write a code that executes but not efficiently, or it can make the programmer more aware of the load balancing or communication issues in a parallel environment which can lead to better parallel algorithms' design [12].

• SPMD and MPMD

In Single Program Multiple Data (SPMD) model, different processes execute the same program on different data. The focus is on performing the same operations on a dataset.

In Multiple Program Multiple Data (MPMD) model, different processes execute different programs on different data.

We have employed both of these models in our parallel designs.

• Decomposition

One of the very first steps in designing a parallel program is deciding on how to divide the work between the processing elements. There are two basic methods to do that: domain/data decomposition and functional decomposition. The former refers to dividing the computational work among multiple processors through distributing the data associated with the problem [13], and the latter is the process of identifying functionally distinct but independent computations [14].

In this research, we have extensively utilized the concept of domain decomposition in our parallel designs.

• Sources of Overhead in Parallel Programs

By using more processes, we expect to have a better performance proportional to the number of processes. But in reality, this almost never happens due to the overheads. Overhead in parallel programs is the lost time which parallel tasks spend on doing different things other than useful work. In parallel computing, we encounter a range of overhead sources. In the following sections, we highlight the major ones which we tried to address in our parallel implementations.

a) Inter-Process Communication

Apart from embarrassingly parallel models (zero communication needed between the processes), parallel processes often need to communicate to perform their task. To improve the parallel processing performance, we need to decrease the frequency and the amount of these communications if and when possible.

b) Idling

Idling happens for a number of reasons such as load imbalance, synchronization, or presence of a serial section in the program. In such situations, parallel resources are not used effectively which leads to poor performance.

c) Load Imbalance

Load imbalance refers to uneven distribution of data or work among processes. Even when we start with evenly distributed data among the processes, we may face the load imbalance issue along the way due to the design of the parallel algorithm and how it proceeds.

- **Amdahl's Law**

Amdahl's law serves as a vital principle in parallel computing, guiding developers in making informed decisions about parallelization strategies and resource allocation. It defines the maximum speed-up that can be achieved by parallelizing a program, taking into account the portion of the code that remains sequential [44].

According to Amdahl's law, the overall speed-up is limited by the proportion of the program that cannot be parallelized, due to the overhead involved in coordinating parallel tasks, communication between processors, I/O, and etc.

The formula is:

$$\text{Speed-up} = \frac{1}{(1-P) + (\frac{P}{N})} \qquad (6)$$

Where P is the proportion of the program that can be parallelized, N is the number of processors, and 1-P is the sequential proportion.

2.6. Summary

In this chapter, we first discussed the most widely used clustering algorithms' classifications alongside their pros and cons. Next, we did a review of the k-modes algorithm which is a type of partitional clustering algorithm developed for categorical data. We then presented the most popular clustering validation metrics to compare the clustering quality produced by different clustering algorithms. Precision, recall, and accuracy were reviewed. We use these metrics to compare the clustering quality of our algorithms with other clustering techniques which will be mentioned in the next chapter.

In the rest of this chapter, we provided a brief overview of distributed memory systems and its widely adopted MPI programming model. Additionally, we reviewed the SPMD and

MPMD parallel computing models, which, alongside MPI, are utilized in our parallel implementations. Also, given the iterative nature and structure of the k-modes algorithm, it is well-suited for parallelization using SPMD and MPI. In Chapter 4, we delve into the implementation specifics of parallelizing the k-modes algorithm.

Chapter 3. Related Work

What have been explored in this thesis can be looked at from different perspectives: sequential categorical data clustering, parallelization of categorical data clustering, and finally clustering categorical data using the ensemble technique.

In this chapter, we first discussed the works related to sequential categorical data clustering from Section 3.1 to Section 3.5. Cao's algorithm [9] of k-modes' initialization is the main algorithm that we adopted to build a new k-modes' initialization method called Revised Density Method (RDM) discussed in chapter 4. We used Ahmad and Dey's [16], He's [19], and also Huang's [10] algorithms for categorical distance measurement in our implementations. Khan's [8] method for identifying the prominent attributes is also utilized in our work for the same purpose. From Section 3.4 to Section 3.5, other state-of-the-art k-modes based clustering algorithms are reviewed briefly. We compared our clustering quality with all their results in chapter 5.

The second half of this chapter is designated to parallel clustering algorithms and an ensemble technique for clustering categorical data. All these algorithms have some features in common with what we have done, either in the type of the data they work with, their clustering algorithm, or even the parallel computing model.

3.1. Cao's Algorithm

Cao [9] introduced a new initialization method to improve the quality of the k-modes clustering algorithm. This method is based on the notation of object's average density and the distance between the objects.

The average density *Dens*(*x*) of the object $x = [x_1, x_2,\ldots, x_m]$ in domain *U* (the non-empty set of objects) is as follows:

$$Dens(x) = \frac{\sum_{a\in A} Dens_a(x)}{m} \qquad (1)$$

Where *m* is the number of attributes, *A* is the set of attributes, and $Dens_a$(x) is the density of object *x* with respect to ath attribute, and is defined as:

$$Dens_a(x) = \frac{f(x_a)}{|U|} \ , \quad 1 \le a \le \text{m} \qquad (2)$$

Where f (x_a) is the frequency of attribute value x_a in domain *U*.

This method starts by choosing the object with the highest average density as the first initial center. The idea is, the denser an object, the better chance it has, to be the centroid of an initial cluster because clearly, it has more data objects around it.

For the remaining centroids, it not only considers the average density of an object, but also the distance between the object and the previously chosen centroids. It aims to choose objects with as much average density as possible, and as far away as possible from the previous centroids. This way, it avoids choosing an object in close vicinity of the previous centers, or an outlier as the next initial center.

Algorithm 1: Initialization method for k-modes algorithm [9]

Input: The dataset *D* and *k*, where *k* is the desired number of clusters.
Output: Centers /centroids

```
1. Centers = $\emptyset$
2. For each $x_i \in U$, calculate Dens($x_i$), Centers = Centers $\cup \{x_i\}$, where $x_i$ satisfies Dens($x_{i_1}$) = $max_{i=1}^{|U|}$ { Dens($x_i$)}, the first cluster center is selected.
3. Find the second cluster center, Centers = Centers $\cup$ {$x_{i_2}$}, where $x_{i_2}$ satisfies d($x_{i_1}, x_m$) $\times$ Dens($x_{i_2}$) = $max_{i=1}^{|U|}$ { d($x_{i_1}, x_m$) $\times$ Dens($x_i$)| $x_m \in$ Centers}, go to step 4.
4. If | Centers | $\leq$ k, then go to step 5, otherwise go to step 6.
5. For any $x_i \in U$, Centers = Centers $\cup$ {$x_{i_3}$}, where $x_{i_3}$ satisfies d($x_{i_3}, x_m$) $\times$ Dens($x_{i_3}$) = max{$min_{x_m}$ { d($x_i, x_m$) $\times$ Dens($x_i$)}| $x_i \in U$},go to step 4.
6. End.
```

As you can see in the lines 3 and 5 of this algorithm, the product d(x_i, x_m) × Dens(x_i) plays a key role in choosing the initial centroids. d(x_i, x_m) is the distance between the objects x_i and x_m, and Dens(x_i) is the density of object x_i. The problem with this approach is that there is no control over the contribution of this product's factors. Even an outlier can be chosen as one of the centroids because it is at a high distance from previous centroids.

We have adopted the Cao's algorithm in our work to solve its initialization issue which we will discuss in Section 4.4.1. In our approach, RDM, the average density of an object is prioritized over its distance from previously chosen centroids. In chapter 5, we compare the clustering quality of RDM and Cao's method in details.

3.2. Khan and Ahmad's Algorithms

In this section, we review two similar and closely related works by Khan's [8] and Ahmad and Dey's [16].

Khan proposes a method of initialization for k-modes by identifying different types of attributes (*Vanilla*, *Prominent*, and *Significant*) in the dataset, and then looking at the data objects from different perspectives created by those attributes [8]. This algorithm generates cluster strings for each data object based on these different perspectives and includes them in the initial clusters if their cluster string is identical or similar. Finally, it calculates the modes of each initial cluster to produce the non-random initial cluster centers (also called centroids) for k-modes algorithm.

- **Vanilla Attributes**

  This method considers all the attributes (m) present in the data and generates M clustering views that will be used in the clustering process [8].

- **Prominent Attributes**

  Prominent attributes are those with less than or equal attribute values to the number of clusters, k. Due to fewer attribute values per cluster, these attributes possess higher discriminatory power and will play a significant role in deciding the initial cluster centers as well as the cluster structures [15].

Algorithm 2: Identification of *Prominent* attributes

Input: D = data objects, n = number of data objects, A = Set of attributes in the data, number of attributes in D: $m = |A|$, p = number of *Prominent attributes* = 0

Output: P = set of *Prominent* attributes

```
1.  $P = \emptyset$
2.  for $i = 1 \rightarrow m$ do
3.      if number of distinct attribute values in $A_i > 1$ && $A_i \leq k$ then
4.          Add $A_i$ to $P$
5.          Increment $p$
6.      end if
7.  end for
8.  if $p = 0 \,||\, p = m$ then
9.      use all attributes as Prominent
10. else
11.     use the set $P$ as the Prominent attributes
12. end if
```

- **Significant Attributes**

  Unlike in numerical data, calculating the distance between two categorical values has posed challenges as there is no straight-forward method. In 2007, Ahmad and Dey [16] proposed an unsupervised learning method to calculate the distance between two distinct values of an attribute. This method is based on the overall distribution of the value pair and their co-occurrence with other attributes.

  After computing the distance between each pair of values belonging to an attribute, they calculate the average of all these distances and assign the resulting numeric value as the significance of the intended attribute. Khan has used their method to identify the most significant attributes based on the number of prominent attributes, that is, if there are *x* prominent attributes, they choose the top *x* most significant attributes based on the calculated significance value.

Algorithm 3: Computing the attributes' significance

Input: *D* = dataset, *n* = number of data objects, *A* = Set of attributes of *D*, and number of attributes in *D*: $m = |A|$

Output: *S* = set of *Significant* attributes, *PD* = set of triplets $(i, (x, y), \Theta)$, where *i* is the number of the attribute, (x, y) is a pair of distinct values of A_i, and Θ is their distance.

```
1.  $S = \emptyset$ , $PD = \emptyset$
2.  for each attribute $A_i$ do
3.      for each pair of categorical attribute values $(x, y)$ do
4.          $Sum = 0$
5.          for every other attribute $A_j$ do
6.              $\theta(i, x, y) = max\ (p\ (w \mid x) + p\ (\sim w \mid y) - 1)$
7.              where $w$ is a subset of j^th attribute values
8.              $Sum = Sum + \theta(i, x, y)$
9.          end for
10.         $\Theta = \frac{Sum}{(m-1)}$, distance $\Theta$ between $x$ and $y$ values of attribute $A_i$
11.     end for
12.     calculate the average of all the pair distances of $A_i$ and assign it as its significance
13.     end for
```

Where:
$p\ (w \mid x)$ is the probability that if attribute A_i's value is *x*, then attribute A_j has values from the subset *w*.
$p\ (\sim w \mid y)$ is the probability that if attribute A_i's value is *y*, then attribute A_j does not have values from the subset *w*.

We have used the concept of prominent attributes in PRM RDM which we are going to discuss in Section 4.4.4. Also, Ahmad and Dey's method of calculating the distance between attribute values has been used in SIG RDM and SIG-SCAV RDM (Section 4.4.5 and Section 4.4.6).

3.3. He's Algorithm on Dissimilarity Measure

When k-modes algorithm was first introduced by Huang [10], a simple hamming distance was used to enable k-means to work with categorical data (Section 2.2, part a). He [19] improved this dissimilarity measure by taking into account the frequency of attribute values of the mode (center) and not only the attribute value of the mode. To clarify this, let us consider the following example:

Data object	A1	A2	A3
$\mathbf{D_1}$	G	M	R
$\mathbf{D_2}$	G	M	H
$\mathbf{D_3}$	G	M	L
$\boldsymbol{Q_1}$	G	M	R

Table 3.1 Cluster C1

Data object	A1	A2	A3
$\mathbf{D_4}$	G	M	F
$\mathbf{D_5}$	X	M	E
$\mathbf{D_6}$	G	B	S
$\boldsymbol{Q_2}$	G	M	F

Table 3.2 Cluster C2

Q_1 and Q_2 are centers of clusters C_1 and C_2.

Now suppose there is a data object X = [G, M, T] which we want to assign to one of these two clusters. For this purpose, we calculate its distance from the cluster centers. Using the hamming distance, we obtain d_1 (*X,* Q_1) = d_1 (*X,* Q_2) = 0 + 0 + 1 = 1, and hence, we cannot decide to which cluster, X should be assigned.

The author extended the hamming distance to solve this problem as follows:

$$d_2\,(X\,,\,Q_l) = \sum_{j=1}^{m} \emptyset(\,x_j\,,\,q_{j,l}) \qquad (3)$$

$$\text{Where } \emptyset(x_j, y_j) = \begin{cases} 1-(|x_{j,l}|/|C_l|), & (x_j\,=\,q_j) \\ 1, & (x_j\,\neq\,q_j) \end{cases}$$

In equation 3:

1. Q_l is the mode of cluster *l*
2. $q_{j,l}$ is the j^th^ attribute value of Q_l.
3. $|x_{j,l}|$ is the number of objects with attribute value equal to x_j in cluster C_l.
4. $|C_l|$ is the number of objects in cluster C_l.

Using the extended distance above, we have:

d_2 (*X,* Q_1) = (1-3/3) + (1-3/3) + 1 = 1

d_2 (*X,* Q_2) = (1-2/3) + (1-2/3) + 1 $\approx$ 1.67

Consequently, with respect to these distances, data object *X* is assigned to cluster C_1.

We used and studied this extended distance measure in some of our initialization algorithms (all RDM variants except SIG RDM and SIG-SCAV RDM).

3.4. K-PbC Algorithm

By looking at a dataset of data objects as a transaction dataset containing items bought by a customer, Dinh and Huynh propose the Pattern Based Clustering (k-PbC) algorithm [7]. K-PbC adapts the Maximal Frequent Itemset Mining (MFIM) algorithm to identify initial non-random clusters. MFIM was proposed to discover association rules between two sets of items, or two data objects as in our context. The method finds the groups of attribute values which frequently co-occur in the datasets [17].

- **FI and MFI**

Frequent itemsets (FIs) are attribute values which occur in a dataset more than a certain number of times called minimum support (*minsup*). Maximal frequent itemsets (MFIs) are FIs that have no frequent superset, that is, MFIs are the largest FIs [7].

- **How it Works**

In the initialization phase of k-PbC, the MFIM algorithm is used to identify the top *k* MFIs from the transaction dataset. For this, it uses the FPMAX algorithm to construct an MFI-tree to find all MFIs. This tree resembles the FP-tree (Frequent Pattern tree) structure which in fact, is the data structure of the popular FP-growth (Frequent Pattern growth) algorithm – an efficient algorithm for mining frequent itemsets using association rules.

To the best of our knowledge, k-PbC is the best available initialization algorithm for k-modes which we have used to compare with our proposed RDM initialization method.

3.5. Other K-Modes Based Clustering Algorithms

In this section, we review six other clustering algorithms briefly. Dinh and Huynh [7] have compared their k-PbC algorithm's performance with them. In our experiments, we also have compared our initialization methods' performance (RDM and RDM Ensemble) with these algorithms.

Two of these six algorithms, i.e., k-means++ and k-means|| work only with numerical datasets. One-hot encoding has been used to allow them to work with categorical datasets [7].

a) K-Means++

This method was invented by Arthur et al. [21] to improve the k-means clustering method through non-random initialization. For the first initial centroid, k-means++ chooses a random object, and then for the rest of the centroids, it selects the data objects with the maximum probability proportional to their squared distance from previously selected centroids, i.e., the centroid with the least distance from the data objects is chosen for this purpose. Having determined the initial centroids, k-means++ then proceeds with the clustering phase similar to the basic k-means algorithm which is the same as the k-modes algorithm

mentioned in Section 2.2 (starting from step 2). The only difference is that instead of re-computing modes, it finds the means of the newly-formed clusters.
One drawback of this method is its sequential approach to selecting *k* centroids which makes it unscalable in dealing with large datasets or datasets with a large number of clusters.

b) K-Means||

Bahmani et al. [22] adapted k-means++ algorithm to solve its scalability issue which achieves approximation guarantees to k-means, i.e., it guarantees to find a solution in polynomial time same as k-means. Then, they parallelized it using MapReduce. K-means|| is faster than k-means++ because it performs fewer iterations for choosing initial centroids by sampling *O* (*k*) data objects as centroid candidates, as opposed to choosing just one, in each iteration using a non-uniform distribution. The number of iterations is almost equal to *O* (log *n*), where *n* is the number of data objects in the dataset.

Using sampling, they choose some data objects as centroid candidates. The algorithm then assigns weights to these objects and reclusters these weighted data objects to obtain *k* centers [22]. The sample set is much smaller than the real dataset. For the purpose of re-clustering, k-means|| uses any provable approximation algorithm such as k-means++ [22].

k-means|| improves both the quality and runtime of k-means.

c) K-Representatives (K-Reps)

In 2004, San et al. [23] proposed a method to cluster categorical data by altering k-means to use random initialization for forming cluster centroids. It replaces addition and multiplication operations with Cartesian product and union operations, in order to form "cluster centers" based on the notion of means in the numerical setting [23]. This algorithm replaces the means in k-means by representatives for each cluster. A representative of cluster *C*, *Q* = $[q_1, q_2, \ldots, q_m]$ is defined as follows:

$$q_j = \{(c_j, f_{c_j}) \mid c_j \in D_j\} \qquad (4)$$

Where m is the number of attributes, D_j is the set formed of categorical values of attribute *j*, and f_{c_j} is the relative frequency of c_j in cluster *C*.

The dissimilarity measure between an object $X_i = [x_{i1}, x_{i2}, \ldots, x_{im}]$ and a representative *Q*, is defined as follows:

$$d\,(I, X) = \sum_{j=1}^{m} \sum_{c_j \in D_j} (f_{c_j} \,.\, \delta(x_j, c_j)) \qquad (5)$$

Where δ is the simple 0,1 dissimilarity measure from Equation 1, Section 2.2.

d) K-Centers, Mod-2, and Mod-3

In 2013, Chen et al. [27] proposed an algorithm similar to k-means called *k-centers* to cluster categorical data which uses a kernel-based method to form cluster centers using a Bayes-type probability estimator. In addition, they use a technique to assign weights to each attribute to measure its individual contribution in the cluster. The distance between cluster centers and objects is estimated by using the simple matching as an indicator function to represent each data object by a set of vectors and the Euclidean norm to quantify the dissimilarity [7].

In 2016, Nguyen et al. [25] changed k-centers algorithm (Mod-3). They did so by using a modified cluster representation based on a kernel density estimate concept of cluster centers for categorical objects. As for the dissimilarity measure, an Information Theoretic Based Dissimilarity measure (ITBD) was used.

In 2019, Nguyen et al. [24] modified k-centers algorithm using Information Theoretic Based Dissimilarity measure (ITBD) instead of the Euclidean norm metric (Mod-2). This ITBD takes the underlying distribution of the categorical attributes into consideration and is used to calculate the distance between the categorical data objects and the cluster centers.

It is worth noting that all these three methods use random initialization, and both Mod-2 and Mod-3 have been proved to outperform k-centers' quality.

e) CD-Clustering

In [26], Nguyen et al. proposed a method called CD-Clustering or Community Detection clustering which tries to solve the issue of k-modes' random initialization. This algorithm uses the Louvain community detection technique to build an unweighted graph and detect highly cohesive groups of nodes. It then chooses the top *k* communities, sorted in descending order by their size, to define *k* modes.

To be able to choose highly cohesive groups or communities, this algorithm uses the hamming distance (Section 2.2) and a threshold *R* which is defined by the number of clusters, *k* and the distance distribution.

One shortcoming of this algorithm is that when the inter/intra clusters' gap and the number of clusters *k* are both small, it cannot detect the communities well [7]. However, the main drawback of Nguyen et al.'s method is its quadratic complexity which limits its application to datasets of 50,000 data objects or less [26].

3.6. Parallel K-Means Using MPI

In [28], Joshi proposed a parallel k-means algorithm using MPI which achieved reasonable speed-ups especially for large datasets. The algorithm first divides the data evenly among all the available processors, and then the master process chooses *k* random initial centroids and broadcasts them to other processes. Each process then enters the phase of calculating the distance between its own data objects and these centroids and decides which of the *k* clusters' centroids is closer to each data object.

At the end of each iteration, two communications must take place: updating the centroids and summing up the Mean Squared Error (MSE) across all the processes.

MSE serves as a termination condition which is satisfied, if it reaches a certain amount or does not change anymore.

In parallel k-means, to re-compute the centroids at the end of each iteration, each process needs to just sum up all the corresponding dimensions of its local data objects in cluster C_i and then, do a collective reduce on this sum and the C_i's local number of data objects. This process is done to calculate the *global mean* for each cluster C_i. As a result, updating the centroids (global means of each cluster) is straight-froward and not challenging as parallel k-modes. The reason lies in the nature of numerical data. The step in which, all the processes sum up their corresponding dimensions is where parallel k-means differs drastically from parallel k-modes. In k-means, a simple sum-up on all the corresponding attribute values' sums is sufficient and can produce the *global sum* across all the processes for each attribute category. Whereas in parallel k-modes, after computing modes for each attribute category in each process *locally*, in order to calculate the *global mode* for each attribute, we cannot simply calculate the *mode of these modes*. The reason is that the collective number of each attribute value in each category can be more than its individual number in one process, and it can affect our decision to calculate the correct modes/centers, to a high extent. So, we need to find a way to share this extra information among the processes while trying not to lose the efficiency of the parallel k-modes. In the next chapter, we propose the PV3 algorithm for this purpose (Section 4.3).

To give an example of the complexity of MPI k-modes, Figure 3.1 is presented. Assuming, we have 3 MPI processes, all holding the data objects of cluster C_1. While the mode of modes in this example is equal to "A" with the overall frequency of 4, the cluster C_1's real global mode across all the processes is "B" with the overall frequency of 6:

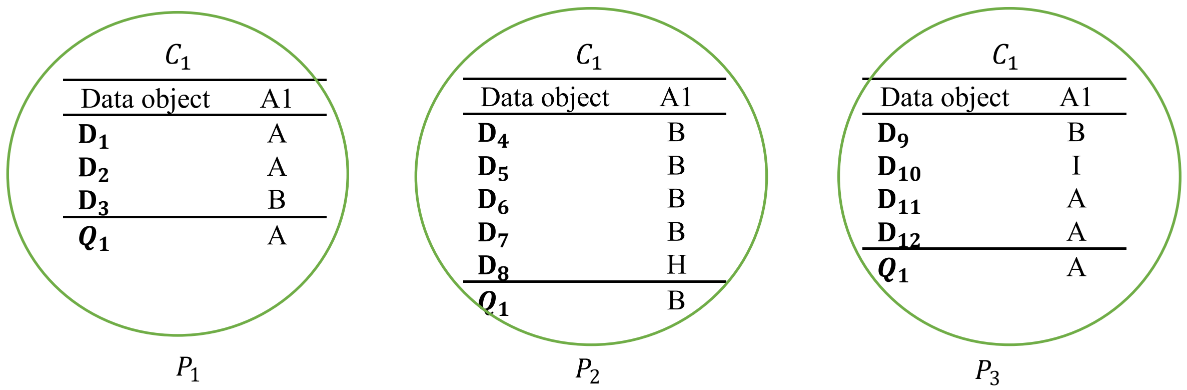

Figure 3.1 Three MPI Processes Holding the Categorical data objects of Cluster C_1

3.7. Parallel K-Means Based on Spark

In [43], Wang et al. propose an improved parallel k-means algorithm based on Spark distributed computing framework. The Canopy clustering algorithm is employed to determine the optimal value of *k*, which represents the number of clusters. Additionally, the selection of initial centroids is performed using the weighted density method to mitigate the influence of outliers on the final clustering outcomes. To measure dissimilarity, a weighted Euclidean distance is utilized. Experimental results obtained from UCI datasets demonstrate that the enhanced k-means algorithm not only enhances the quality of clustering but also reduces the average iteration count. Moreover, the parallel computing performance is evaluated in a Spark cluster environment with multiple nodes, revealing a reduction in the algorithm's execution time.

3.8. Parallel K-Modes Based on MapReduce

Tao et al. [29] implemented k-modes algorithm on Hadoop using MapReduce parallel computing model. However, their implementation focused on the original k-modes algorithm without addressing its initialization issue, which significantly affects the quality of clustering results.

They improved the process of k-modes: when allocating categorical objects to clusters, the number of each attribute value in clusters is updated, so that the new modes of clusters can be computed after reading the whole dataset once [29].

Step 1. Firstly, *k* random data objects are chosen to serve as the initial centroids. Then, *p* map tasks are defined, and the data is divided between them evenly. Each map task stores the data objects in the form of key-value pairs as <cluster_id, object>.

Step 2. In the next step, each map calculates the distance between each object and centroid to decide the new memberships and updates the cluster id for each data object. In addition, the number of each attribute value in clusters are updated to avoid reading the data one more time for the purpose of re-computing the centroids.

Step 3. Once the new modes/centers are calculated by reading the clusters' information written by *p* maps and choosing the item with the maximum frequency, the distance between the new and the old modes is measured. If this distance is more than some threshold, the algorithm returns to step 2. Otherwise, it goes to step 4.

Step 4. *k* reduce tasks are defined which generate *k* clusters in the output files using the information from the *p* maps.

The algorithm showed good speed-ups when clustering large datasets [29].

3.9. A Cluster Ensemble Framework for Categorical Data

In [30], He et al. showed that cluster ensemble and clustering categorical data are the same problem due to the extensive commonalities between them. In other words, clustering categorical data is an optimization problem from the viewpoint of cluster ensemble [30].

Their ensemble algorithm (ccdByEnsemble) has used three hypergraph-model based algorithms namely CSPA, HGPA, and MCLA, adapted from Strehl et al. [33, 34].

- *CSPA*: cluster-based similarity partitioning algorithm is simply evolved around an $n \times n$ binary similarity matrix in which, “1” indicates the two objects are in the same cluster, and “0” means they are in different clusters. From this similarity matrix, a similarity graph is generated. They then use the METIS algorithm [35] to partition this graph into clusters.

- *HGPA*: In this Algorithm, each cluster is represented by a hyperedge with the same weights, and the data objects are the vertices with the same weights too. HMETIS algorithm [36] is then used to partition this hypergraph such that the sum of weights the hyperedge cut is minimized [30].

- *MCLA*: Similar to HGPA, in this algorithm, each cluster is represented as a hyper-edge. It starts by forming hyper-edges which are more than the number of clusters, based on some relations [34]. Then, in the next stages, the algorithm groups and collapses related hyper-edges. This process terminates until there are *k* clusters left.

He et al. evaluated their cluster ensemble algorithm on 4 UCI datasets (Vote, Breast cancer, Zoo, and Mushroom) using clustering accuracy measure. They employed the aforementioned three methods and selected the best results for each dataset.

As we will see later in section 4.4.7, the accuracy of our RDM Ensemble is significantly higher (20%) for the Mushroom and Zoo datasets, and it is less accurate for the datasets Breast cancer (6% poorer) and Vote (2% poorer).

Chapter 4. Proposed Solution Techniques

In this chapter, we first present ideas of different designs and developments which we explored for parallelizing the k-modes algorithm. These attempts led us to our proposed solution, the PV3 algorithm. In our work, we considered the Single Program Multiple Data (SPMD) using Message Passing Interface (MPI) on a distributed memory platform. The three solution versions developed, include PV1, PV2, and PV3, where the first two paved the way for the final solution, PV3. Section 4.1 to Section 4.3 provide details of them.

In Section 4.4, we introduce RDM which we adopted from the Cao's algorithm for the initialization step of our solution. We then incorporated this adopted RDM in various initialization solutions to improve it and developed different RDM variants.

In Section 4.5, through parallelizing RDM and plugging it in PV3, PV3R is designed. In Section 4.6, we present details of parallelization of another RDM variant (SIG RDM), and finally in Section 4.7, we propose a solution framework called Ensemble Parallelizing Process (EPP) for categorical data clustering parallelization. We then use all our RDM variants and the PV3 algorithm to build a realization of EPP to produce RDM ensemble clustering result.

In this thesis, the projects were implemented using the C++ programming language with the GNU C++ Compiler (version 11.3.0). Also, the implementation leveraged the parallel computing capabilities provided by Open MPI (version 4.1.2) for efficient distributed processing.

4.1. Parallel Algorithm Version 1 (PV1)

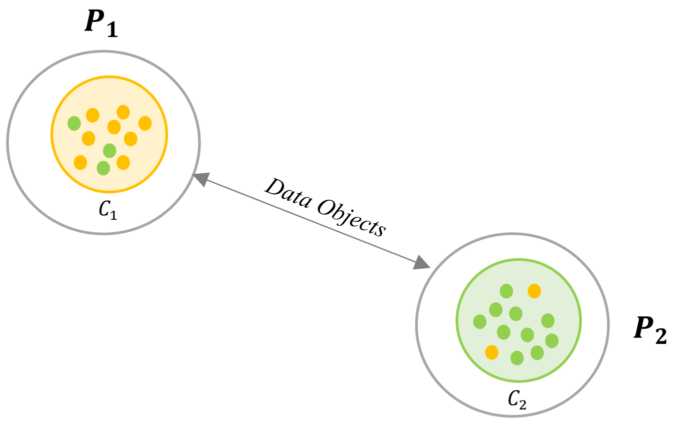

Figure 4.1 PV1's Schema for k = 2 and p = 2

Our first goal was parallelizing the k-modes clustering algorithm using MPI. An intuitive solution was to view each cluster as a *process* (processing element) responsible for the computations of that cluster. As the standard sequential k-modes clustering is an iterative process in which data are exchanged among the clusters at the end of each iteration, the view adopted above induces that the corresponding processes perform exchanging data at the end of each iteration to synchronize (Figure 4.1). The advantages and disadvantages of this view are as follows.

The advantages are:

1. Being easy and straight-forward to implement.
2. Having a real image of what is happening in other processes due to the fact that we are transferring data objects among the processes or clusters.
3. Obtaining the same quality of sequential algorithm in parallel environment which means if we can find a way to improve the quality of the sequential algorithm, we can easily reproduce its clustering results using the parallel version algorithm.

The disadvantages are:

1. We should use only the number of processes equal to the number of clusters, and hence, any additional resources available would remain unused.
2. Load imbalance is more likely to happen in this version, especially when we have uneven distribution of the data objects in the real clusters.
3. High communication cost. Since data objects are being exchanged among the processes in each iteration, the communication cost could be very high, hence, resulting in poor performance.

Algorithm 1: Parallel Version 1 (PV1)

Input: D = data objects, n = number of data objects, k = number of clusters, p = number of processes: P_i, $i \in \{1, \ldots, k\}$

Output: k clusters of D

```
1.  $E\_global_0$ = -1
2.  for each $P_i$ , $i \in \{1, \ldots, k\}$ do
3.      read ($d_i$): $\frac{n}{k}$ distinct data objects from D.
4.  end for
5.  All processes collectively sum all the $E\_local_1$ up and store it in $E\_global_1$.
6.  while $E\_global_1 \neq E\_global_0$ do
7.      for each $P_i$ , $i \in \{1, \ldots, k\}$ do
8.          $m_i$ = find_mode ($d_i$)
9.          send $m_i$ to other processes and receive other processes' modes to build M = $\{m_1, \ldots, m_k\}$
10.         re_cluster ($d_i$, M)
11.         send/receive the farthest/closest data objects to/from other processes after reclustering
            based on the closest_cluster_index produced by function re_cluster.
12.         $E\_local_1$ = E_local_calculator_single ($d_i$, $m_i$)
13.     end for
14.     $E\_global_0 = E\_global_1$
15.     All processes collectively sum all the $E\_local_1$ up and store it in $E\_global_1$.
16. end while
17. each process $P_i$ produces its corresponding $C_i$ and stops.
```

In line 8, for the function *find_mode*, we simply find the most frequent attribute value of each attribute category in each process's share of data.

In line 10, function *re_cluster* is used to find the distances between the data objects and the new centroids in each iteration in order to find the closest centroid for each object:

Function 1: re_cluster

Input: d_i = each P_i's share of data, M = set of all cluster modes

Output: *closest_cluster_index* array with $\frac{n}{p}$ indices for objects in d_i

```
1.  min_dist = max_float
2.  for each $x_j \in d_i$ do
3.      for each $m_l \in M$ do
4.          $d_{jl}$ = find_distance ( $x_j$, $m_l$)
5.          if $d_{jl}$ < min_dist then
6.              min_dist = $d_{jl}$
7.              closest_cluster_index[j] = l
8.          end if
9.      end for
10. end for
```

In line 4 of Function 1, for the function *find_distance*, we have used the He's distance measure of Section 3.3.

Function 2: E_local_calculator_single

Input: d_i = each P_i's share of data, m_i = each process/cluster's mode

Output: *E_local*: sum of all distances between each data object x_j and its corresponding mode m_i.

```
1. E_local = 0
2. for each $x_j \in d_i$ do
3.     E_local = E_local + find_distance ($x_j$, $m_i$)
4. end for
```

4.2. Parallel Algorithm Version 2 (PV2)

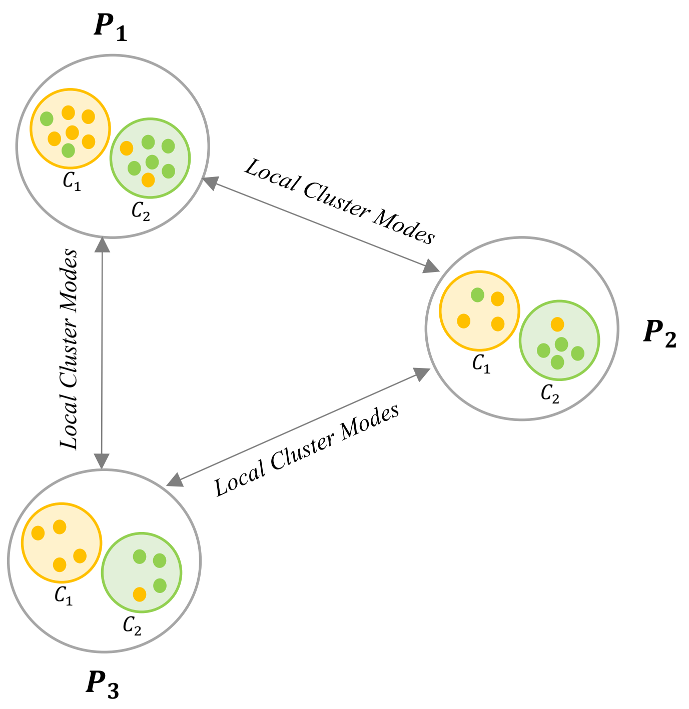

Figure 4.2 PV2's Schema for k = 2 and p = 3

As mentioned earlier, the main shortcoming of the PV1 algorithm was being bound to the number of clusters when employing processes. It is a big issue, especially when it comes to large datasets because we cannot use our parallel resources in an efficient way. To support this view, we designed algorithm Parallel Version 2 (PV2) which we view as a stepping stone to develop Parallel Version 3 (PV3), details of which will be provided in the following section.

Unlike PV1, in PV2, we can use as many processes as desired, and each process may also hold data objects from other clusters. PV2 starts by assuming that each processing element has data objects which belong to all clusters and divides its own share of data into *k* distinct clusters. In other words, each process clusters its share of data as done in a single sequential version, and once each processing node completes an iteration step, the local cluster modes of processes are exchanged among them before starting the next iteration step (Figure 4.2).

The advantages of this approach are:

1. Load balance. Each process's load balance is going to stay the same throughout the whole clustering process because the data objects are not exchanged unlike in PV1.
2. Fast. Due to the fact that the amount of exchanged data in each iteration is small, this version is fast.
3. Efficient use of the resources. Since we are not bound to the number of clusters in using processes, we are free, in principle, to use as many processes as desired.

Clearly, the main drawback of PV2 is lack of exchanged information among the processes, and by increasing the number of employed processes, the data becomes more fragmented (since only local modes are exchanged between the processes). This leads to poor clustering result in particular when using more processes. This issue undermines the whole purpose of this version which was removing the limitation on the number of processes. The steps of PV2 is described in Algorithm 4, as follows:

Algorithm 2: Parallel Version 2 (PV2)

Input: D = data objects, n = number of data objects, k = number of clusters, p = number of processes P_i, $i \in \{1, \ldots, p\}$

Output: k clusters of D

```
 1. $E\_global_0$ = -1
 2. for each $P_i$ , $i \in \{1, \ldots, p\}$ do
 3.     read ($d_i$): $\frac{n}{p}$ distinct data objects from D with the size $n_i$.
 4.     randomly divide $d_i$ into k local clusters with equal size of $\frac{n_i}{k}$ to build $C_i$ = { $C_{i1}$,…,$C_{ik}$}.
 5. end for
 6. All processes collectively sum all the $E\_local_1$ up and store it in $E\_global_1$.
 7. while $E\_global_1 \neq E\_global_0$ do
 8.         for each $P_i$ , $i \in \{1, \ldots, p\}$ do
 9.             $M_i$=find_modes ($C_i$), $M_i$ = {$m_{i1}$,…, $m_{ik}$}
10.             send $M_i$ to other processes and receive other processes' $M_i$
11.             calculate mode of modes, M = {$m_1$,…, $m_k$}, using all the $M_i$s of all processes.
12.             re_cluster ($C_i$, M)
13.             move objects locally between clusters based on their new cluster indices.
14.             $E\_local_1$= E_local_calculator_modes ($C_i$, M)
15.         end for
16.         $E\_global_0 = E\_global_1$
17.         All processes collectively sum all the $E\_local_1$ up and store it in $E\_global_1$.
18. end while
19. each process $P_i$ sends its k clusters to Master ($P_0$) and stops.
20. Master outputs the integrated clusters.
```

In line 9, function *find_modes* is used to find the most frequent values of each attribute category in all the *k* clusters of a process's share of data d_i.

A description of function *re_cluster* in line 12 is given in Function 1 of Section 4.1.

In line 14, *E_local_calculator_modes* is the cost function which sums up all the distances between the data objects in a cluster and their corresponding centroid as follows:

Function 3: E_local_calculator_modes

Input: k local clusters of each process: $C = \{C_1,\dots,C_k\}$, Mode of modes of all clusters in all the processes: M = $\{m_1, \dots, m_k\}$

Output: *E_local*: sum of all the distances between each data object x_j and its corresponding mode m_i.

```
1. sum_E_local = 0
2. for each $m_i$ ∈ M, i ∈ {1, …, k}
3.     for each $x_j \in C_i$
4.         sum_E_local = sum_E_local + E_local_calculator_single ($x_j$, $m_i$)
5.     end for
6. end for
```

4.3. Parallel Algorithm Version 3 (PV3)

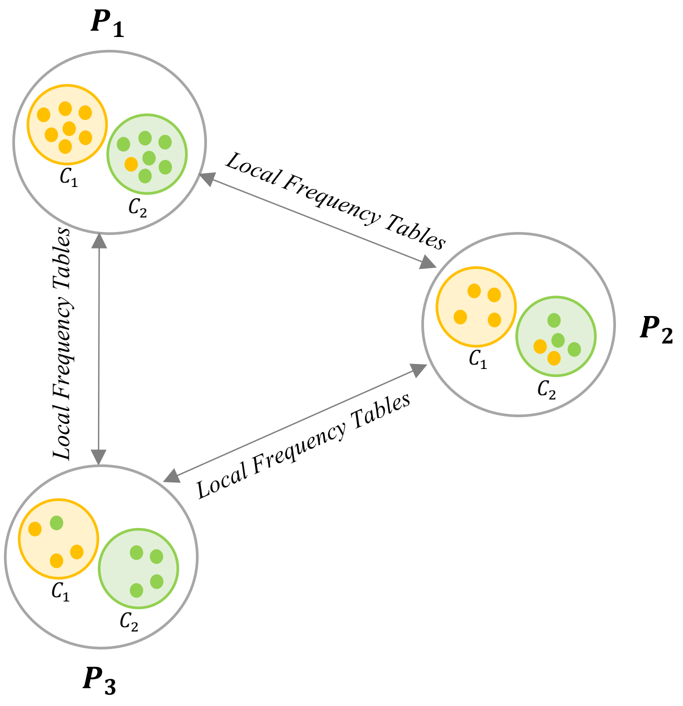

Figure 4.3 PV3's Schema for k = 2 and p = 3

As mentioned, in PV1, we had a limitation on the number of processes. To solve that, we designed PV2 but at the cost of losing quality especially when using more processes. Our attempts to overcome the issues of PV1 and PV2 brought us to the design of the parallel algorithm version

3, PV3. In this version, we can use as many processes as we want unlike in PV1 but not at the cost of losing quality unlike in PV2.

PV3 works the same as PV2 but the only difference is in the amount of exchanged information. At the end of each k-modes' iteration, instead of exchanging local modes, every process exchanges its frequency table which, in fact, is a full summary of the data objects it is holding (Figure 4.3). This frequency table consists of the number of each attribute value of each attribute category in each cluster. For example, Table 4.1 is a categorical dataset with three attributes and 5 objects in two clusters C_1 and C_2:

Cluster name	Data object	Attibute1	Attribue2	Attribute3
$\boldsymbol{C_1}$	D_1	H	C	E
$\boldsymbol{C_1}$	D_2	H	C	F
$\boldsymbol{C_1}$	D_3	T	C	F
$\boldsymbol{C_2}$	D_4	T	K	F
$\boldsymbol{C_2}$	D_5	T	D	F

Table 4.1 Categorical Dataset

The frequency tables for the two clusters in Table 4.1 are as follows:

$\boldsymbol{C_1}$:

Attribute1	H	T
	2	1

Attribute2	C
	3

Attribute3	E	F
	1	2

Table 4.2 Frequency Table of C1

$\boldsymbol{C_2}$:

Attribute1	T
	2

Attribute2	K	D
	1	1

Attribute3	F
	2

Table 4.3 Frequency Table of C2

PV3 is basically a middle ground between PV1 and PV2. Similar to PV1, we do not lose any data while we are not slow (only a summary of what each process has, is exchanged, not the data objects themselves). Similar to PV2, we can use as many processes as we would like but not at the cost of information loss.

Algorithm 3: Parallel Version 3 (PV3)

Input: *D* = data objects, *n* = number of data objects, *k* = number of clusters, *p* = number of processes P_i, $i \in \{1, \ldots, p\}$

Output: *k* clusters of *D*

```
1.  $E\_global_0$ = -1
2.  for each $P_i$ , $i \in \{1, \ldots, p\}$ do
3.      read ($d_i$): $\frac{n}{p}$ distinct data objects from D.
4.      randomly divide $d_i$ into k local clusters with equal size of $\frac{n_i}{k}$ to build $C_i = \{ C_{i1},\ldots,C_{ik}\}$.
5.  end for
6.  All processes collectively sum all the $E\_local_1$ up and store it in $E\_global_1$.
7.  while $E\_global_1 \neq E\_global_0$ do
8.      for each $P_i$ , $i \in \{1, \ldots, p\}$ do
9.          $f_i$= frequency_table_builder ($C_i$)
10.         send $f_i$ to other processes and receive other processes' $f_i$.
11.         integrate all the $f_i$s to build the global frequency table F.
12.         calculate real global modes M = $\{m_1,\ldots, m_k\}$ from global F.
13.         re_cluster ($C_i$, M)
14.         move objects locally between clusters based on their new cluster indices.
15.         $E\_local_1$= E_local_calculator_modes ($C_i$, M)
16.     end for
17.     $E\_global_0 = E\_global_1$
18.     All processes collectively sum all the $E\_local_1$ up and store it in $E\_global_1$.
19. end while
20. each process $P_i$ sends its k clusters to Master ($P_0$) and stops.
21. Master outputs the integrated clusters.
```

Function *re_cluster* in line 13 is defined in Function 1 of Section 4.1.

In line 9, the function *frequency_table_builder* builds the frequency table of each process as follows:

Function 4: frequency_table_builder

Input: k local clusters of each process: $C = \{C_1,\ldots,C_k\}$, A = Set of attributes in the data, number of attributes in the dataset: $m = |A|$

Output: frequency table of each process = f

```
1. for each $C_i \in C$ do
2.     for each $A_j, j \in \{1, \ldots, m\}$ do
3.         for each $X_l \in C_i$ do
4.             find the count of all distinct values of $A_i$ and add it to $f$
5.         end for
6.     end for
7. end for
```

4.4. RDM Algorithm and Variants

After several attempts to parallelize the k-modes algorithm, we noted an issue which is inherent to the nature of any partitional algorithm such as k-modes. Even for PV1 and PV3, in which no information loss is guaranteed, our results are non-deterministic and suffer from the random initialization phase of the k-modes algorithm. This means, the order of the input data objects which affects the initial seeding of the k-modes algorithm can change the resulting clusters drastically in each run of the algorithm.

Numerous studies have been conducted to solve the random initialization issue of the k-modes algorithm over the years [7, 8, 9, 26, 39, 40]. We studied and used some of them to build the RDM initialization method and its variants. The Cao's algorithm [9] is adopted to build RDM which its details are presented in the next section. The method to identify prominent attributes in categorical data objects [15] is used to build a variant of RDM called PRM RDM in Section 4.4.4. Ahmad's algorithm [16] for calculating distance between unique value pairs of a categorical attribute is incorporated to build SIG RDM and SIG SCAV RDM in Sections 4.4.5 and 4.4.6 respectively.

In Section 3.1**,** we reviewed Cao's algorithm [9] in which the author introduced a method for initialization in the k-modes algorithm based on the density of data objects and their distance. We have adopted Cao's algorithm in our work to improve clustering quality of the k-modes.

In Cao's initialization method described as Algorithm 1 of Section 3.1, it first chooses the object with the highest density as the first initial center using Equation 1. As for the second centroid, the object with the maximum density and distance from the first centroid is chosen. This is done by calculating the product (multiplication) of the density of the object and its distance from the first centroid and then, finding the object with the maximum value of this product. If $k > 2$ (number of clusters), for the remaining initial centroid(s), Cao uses a Max-Min approach to decide which object(s) is/are more fitted. This is done also by calculating the product of the object's density and its distance from the previous centroids. In other words, the main criterion and the beating heart of Cao's method is calculating this multiplication:

Distance from previous centroid(s) × *Density of the object*

One problem with this approach is that if the distance factor is much higher than the density, the density can be overwritten by the distance, and as a result it may negatively affect the decision about the initial clusters. For example, if there is an object $\boldsymbol{x}$ with low density which is very distant from our previous centroids (such as an outlier), the algorithm fails to realize and may choose $\boldsymbol{x}$ as the next initial centroid.

To solve this issue, we have adopted Cao's algorithm and proposed the Revised Density Method (RDM) method introduced in the next section.

4.4.1. RDM Initialization Method

Our very first attempt to solve the discussed issue of Cao's method [9] was RDM. This algorithm tries to prioritize the density factor, i.e., instead of calculating the product of density and distance for which we have no control over the contribution of its factors, in each iteration, we choose the highest density object as our centroid and then calculate its distance from all the other remaining objects. We then take the average of all these distances and put the objects whose distances are at most this average in the cluster of the corresponding centroid. The goal here is to form primary dense clusters around our core objects (the centroids) to help us identify better or real initial clusters.

Algorithm 4: RDM Initialization

Input: *D* = data objects = $\{X_1, X_2, \ldots, X_n\}$, *n* = number of data objects, *k* = number of clusters

Output: *k* initial clusters $C_1, C_2, \ldots, C_k$

```
1.  calculate the density for each data object and put the pair (Density, Object) in Density-Object_List
    sorted in descending order by density with size n.
2.  for i = 1 → k do
3.      $Q_i$ = Density-Object_List [0]          //Object with highest density as centroid $Q_i$.
4.      sum = 0
5.      for each $X_j \in$ Density-Object_List do
6.          distance = find_distance ($X_j$, $Q_i$)
7.          sum = sum + distance
8.      end for
9.      ave_dist = $sum/n - 1$
10.     for each $X_j \in$ Density-Object_List do
11.         distance = find_distance ($Q_i$, $X_j$)
12.         if distance $\leq$ ave_dist then
13.             add $X_j$ to $C_i$ and remove it from Density-Object_List.
```

```
14.             $n = n - 1$
15.         end for
16.     end for
17.     if Density-Object_List is not empty then
18.         calculate the modes of all the newly formed initial clusters and assign the remaining objects to
            the closest cluster based on their distance from the clusters' modes.
19.     produce the final $k$ initial clusters $C_1$, $C_2$,…, $C_k$.
```

For details of calculating the density of each object in line 1, please refer to Equations 1 and 2 in Section 3.1.

In line 6, for the function *find_distance*, we used the hamming distance described in part a of Section 2.2.

In line 12, to find the distance between the modes and the remaining unassigned objects, we used the He's distance measure mentioned in Section 3.3. Finally, the *Density-Object_List* in line 1 is a map between the data objects and their density which are sorted based on the density values in descending order:

$Object_1$	$Density_1$
$Object_2$	$Density_2$
. . .	. . .
$Object_{n-1}$	$Density_{n-1}$
$Object_n$	$Density_n$

Table 4.4 Density-Object_List in the First Iteration

As can be seen in the algorithm of RDM, by choosing the highest density object in each iteration (and not just in the first step as done in Cao's method) we have prioritized the object density over the distance measure.

We have used 15 benchmark datasets from the UCI Machine Learning Repository [22] to evaluate the performance of our algorithms. The results of our experiments (Tables 5.6, 5.7, and 5.8 in Section 5.3.1) indicate that RDM improves the clustering quality for almost 60% of these datasets in terms of accuracy, precision, and recall. For one dataset (Soybean), the performance result remained the same (with 100% accuracy, precision, and recall). For the rest, Cao's algorithm performed better.

We explored different solution ideas and techniques to improve RDM. In what follows, we present the ones that showed more promise to produce better results.

4.4.2. GLMD RDM Initialization

In the first modification of RDM, called Global Mode RDM (GLMD RDM), the goal was to eliminate the chance of a border data object to be selected as a centroid. Since in each iteration, the highest density object is chosen as the next centroid, a border object may be chosen [40]. We tried to solve this issue by changing the method for choosing the centroids (Q_is) in each round, as follows. Instead of only assigning the highest density data object as the centroid of our next cluster as in RDM, we calculate the global mode of all the current data objects present in each iteration and then among a list of candidates for the next centroid (a set of high-density objects), we choose the object with the maximum distance from the current global mode. For the list of candidates, we use a heuristic approach. First, we examine our *Density-Object_List* to find the approximate position of the first big density gap. After recognizing at which index, this gap happens, we put all the data objects before the gap index in the candidate list for the next centroid. In the next step, we choose the farthest candidate from the current global mode as our centroid.

Algorithm 5: GLMD RDM Initialization

Input: *D* = data objects = {X_1 , X_2,…, X_n }, *n* = number of data objects, *k* = number of clusters

Output: *k* initial clusters C_1, C_2,…, C_k

```
1.  calculate the density for each data object and put the pair (Density, Object) in Density-Object_List
    sorted in descending order by density with size n.
2.  for i = 1 → k do
3.      find the mode of Density-Object_List as the Current_global_mode.
4.      idxOfFirstDensGap = find_first_gap_index (Density-Object_List).
5.      go through the Density-Object_List and put all the objects before the idxOfFirstDensGap in the
        Candidates_list.
6.      max_distance = 0
7.      for each $X_j$ ∈ Candidates_list do
8.          distance = find_distance ($X_j$, Current_global_mode)
9.          if distance > max_distance then
10.             max_distance = distance
11.             $Q_i$ = $X_j$
12.     end for
13.     sum = 0
14.     for each $X_j$ ∈ Density-Object_List do
15.         distance = find_distance ($X_j$, $Q_i$)
16.         sum = sum + distance
17.     end for
```

```
18.     $ave\_dist = sum/n - 1$
19.     **for each** $X_j$ ∈ *Density-Object_List* **do**
20.         *distance* = *find_distance* ($Q_i$, $X_j$)
21.         **if** *distance* ≤ *ave_dist* **then**
22.             add $X_j$ to $C_i$ and remove it from *Density-Object_List.*
23.             $n = n - 1$
24.     **end for**
25. **end for**
26. **if** *Density-Object_List* **is not** empty **then**
27.     calculate the modes of all the newly formed initial clusters and assign the remaining objects to
        the closest cluster based on their distance from the clusters' modes.
28. produce the final *k* initial clusters $C_1$, $C_2$,…, $C_k$ and stop.
```

In line 4, the function *find_first_gap_index* finds the first large density gap in the *Density-Object_List* as follows:

Function 5: find_first_gap_index

Input: *Density-Object_List, n = Density-Object_List*'s size

Output: *idxOfFirstDensGap*

```
1.  sum = 0
2.  **for** $i = 1 \rightarrow (n - 1)$ **do**
3.      gap = Density-Object_List[i] - Density-Object_List[i+1]
4.      add gap to Gaps array
5.      sum = sum + gap
6.  **end for**
7.  $ave\_gap = sum/n - 1$
8.  idxOfFirstDensGap = n
9.  **for** $i = 1 \rightarrow (n - 1)$
10.     **if** Gaps[i] > ave_gap **then**
11.         idxOfFirstDensGap = i
12.         break
13. **end for**
```

The GLMD RDM's clustering quality results in terms of accuracy, precision, and recall are presented in Tables 5.9, 5.10, and 5.11 in Section 5.3.2.

4.4.3. SCAV RDM Initialization

second average RDM or SCAV was our next attempt to improve RDM by choosing fewer but closer data objects to the centroids. With this approach, we hoped to form smaller but denser neighborhoods around the centroids to enable them to attract more similar data objects into their clusters in the following stages of the algorithm.

In RDM, in each iteration, we calculate the average distance between the centroid and all the data objects still present in our *Density-Object* list. Next, to form initial clusters around the centroid, we put, in the corresponding cluster, all the data objects whose distance is at most this average. Let us call the set of these objects *M*. At this stage, RDM is finished with forming one cluster and continues to find the next one, if any. But as for the SCAV RDM, it does not simply put the objects present in *M*, in the cluster. This algorithm calculates a second average distance on all of *M*'s data objects and then puts all the objects whose distance is at most this *second average* in the corresponding cluster. The steps are formally presented in the following algorithm:

Algorithm 6: SCAV RDM Initialization

Input: *D* = data objects = {X_1 , X_2,…, X_n }, *n* = number of data objects, *k* = number of clusters

Output: *k* initial clusters C_1, C_2,…, C_k

```
1.  calculate the density for each data object and put the pair (Density, Object) in Density-Object_List
    sorted in descending order by density with size n.
2.  for i = 1 → k do
3.      $Q_i$ = Density-Object_List [0]            //Object with highest density as centroid $Q_i$.
4.      $sum_1$ = 0
5.      for each $X_j$ ∈ Density-Object_List do
6.          distance = find_distance ($X_j$, $Q_i$)
7.          add the distance and its corresponding object $X_j$ to Distance-Object_List.
8.          $sum_1$ = $sum_1$ + distance
9.      end for
10.     ave_dist = $sum_1/n - 1$
11.     for each $X_j$ ∈ Distance-Object_List do
12.         distance = Distance-Object_List [$X_j$]
13.         $sum_2$ = 0
14.         m = 0      //m is the size of set M.
15.         if distance ≤ ave_dist then
16.             $sum_2$= $sum_2$ + distance
17.             add $X_j$ to the set M.
18.             m = m + 1
19.     end for
20.     sec_ave_dist = $sum_2/m$ - 1
21.     for each $X_j$ ∈ M do
```

```
22.           distance = find_distance ($X_j$, $Q_i$)
23.           if distance ≤ sec_ave_dist then
24.             add $X_j$ to $C_i$ and remove it from Density-Object_List.
25.             $n = n - 1$
26.       end for
27. end for
28. if Density-Object_List is not empty then
29.   calculate the modes of all the newly formed initial clusters and assign the remaining objects to
      the closest cluster based on their distance from the clusters' modes.
30. produce the final k initial clusters $C_1$, $C_2$,…, $C_k$ and stop.
```

Our evaluation results in terms of accuracy, precision, and recall are shown in Tables 5.9, 5.10, and 5.11 in Section 5.3.2.

4.4.4. PRM RDM Initialization

Prominent RDM (PRM RDM) was our next endeavor to improve RDM by trying to improve the distance measure of RDM. We used the prominent attribute concept mentioned in Section 3.2 for calculating the pair-wise distance between data objects. The idea here was the fact that prominent attributes (attributes with values less than or equal to the number of clusters) have more discriminatory power over other attributes because of their nature [15].

In this method, the indices of the prominent attributes are first identified, but instead of including all the attributes in the distance calculation, we only use the prominent ones. In the cases where the attributes are all prominent or none are so, it performs like the old way, and Hamming distance is used for all the attribute values of two objects.

The pseudo-code for the PRM RDM algorithm is exactly the same as that of RDM provided in Section 4.4.1 except for function *find_distance* in line 6 which is replaced by *find_distance_prm* defined as follows:

Function 6: find_distance_prm

Input: *A* = set of attributes in the data, number of attributes: $m = |A|$, P = set of prominent attributes' indices, first data object: $X_1 = \{x_{11}, x_{12},\ldots, x_{1m}\}$, second data object: $X_2 = \{x_{21}, x_{22},\ldots, x_{2m}\}$

Output: *Distance* between X_1 and X_2.

```
1. Distance = 0
2. for $i = 1 \rightarrow m$ do
3.     if $i \in P$ then
4.         if $x_{1i} \neq x_{2i}$ then
5.             Distance = Distance + 1
6. end for
```

The PRM RDM method's results in terms of accuracy, precision, and recall are presented in Tables 5.9, 5.10, and 5.11 in Section 5.3.2.

4.4.5. SIG RDM Initialization

The distance measurement in the prominent RDM method did not work very well. So, we looked for better ways to measure distance between the data objects. We used Ahmad and Dey's method [16] to calculate the distance between each value pair of each attribute. Significance RDM (SIG RDM) works the same as RDM but instead of using the function *find_*distance in line 6, it uses the triplets produced by Algorithm 3 of Section 3.2 to find the unique distance for each pair of attribute values.

The SIG RDM method's clustering quality results are presented in Tables 5.9, 5.10, and 5.11 in Section 5.3.2.

4.4.6. SIG-SCAV RDM Initialization

In this method, we combined the SCAV algorithm (Section 4.4.3) with the SIG algorithm. So, for distance measurement, we use the significance method mentioned in previous section but similar to the SCAV method, we do not stop at the first average distance. We proceed further to calculate a second one and then form the initial clusters around our chosen centroids.

The SIG-SCAV RDM method's clustering quality results are presented in Tables 5.9, 5.10, and 5.11 in Section 5.3.2.

4.5. Parallel RDM Initialization and Parallel V3R (PV3R)

After exploring ways to improve clustering quality, and producing repeatable and deterministic clustering results using the RDM and its different variants' initialization techniques, we used these produced initial clusters by RDM in PV3. The same clustering results as in the sequential k-modes with RDM initialization were produced in parallel environment. So, while we were able to maintain the same clustering quality, we could benefit from running the program in parallel. So, if we use better initial clusters by RDM or any other state-of-the-art clustering initialization methods for k-modes, we can expect the same clustering quality produced by PV3 as in the sequential version.

we then parallelized RDM's initialization process which helped us reach up to 2.4 times speed-up using 8 processes in the initialization phase. Although the sequential RDM algorithm is fairly simple and fast, and the gained speed-up of its parallelization was not very significant, its parallelizing process allowed us to build the parallel mainframe for other RDM variants such as SIG RDM which is the most time-consuming RDM variant.

The parallel RDM's steps are formally presented in the following algorithm:

Algorithm 7: Parallel RDM initialization

Input: D = data objects, n = number of data objects, k = number of clusters, p = number of processes P_i, $i \in \{1, \dots, p\}$

Output: k initial clusters of D

```
1.  for each $P_i$ , $i \in \{1, \dots, p\}$ do
2.      read ($d_i$): $\frac{n}{p}$ distinct data objects from D.
3.  end for
4.  for each $P_i$ , $i \in \{1, \dots, p\}$ do
5.      $f_i$_nc = frequency_table_nc_builder ($d_i$)
6.      send $f_i$_nc to other processes and receive other processes' $f_i$_nc
7.      integrate all the $f_i$_nc from all processes to build the global frequency table F_nc.
8.      calculate the density for each object based on F_nc and put the pair of (Density, Object) in
        Density-Object_List_Local sorted in descending order by density.
9.  end for
11. for each $P_i$ , $i \in \{1, \dots, p\}$ do
12.     for $m = 1 \rightarrow k$ do
13.         if $P_i$ has still data objects left in Density-Object_List_Local then
14.             $myCandidate_i$ = Density-Object_List_Local [0]        // Object with highest density in $P_i$.
15.             send $myCandidate_i$ to other processes and receive other processes' candidates.
16.             choose the candidate with maximum density as the next centroid $Q_m$.
17.             sum_local = 0
18.             for each $X_j \in$ Density-Object_List_Local do
19.                 distance = find_distance ($X_j$, $Q_m$)
```

```
20.                 sum_local = sum_local + distance
21.             end for
22.             send my sum_local to other processes and receive other processes' sum_local.
23.             send the current size of my available objects as $n_m$ to other processes' and receive other
                processes' $n_m$s.
24.             sum all the sum_locals to build sum_global.
25.             sum all the $n_m$s to build the total size of all the available data objects in iteration m or $N_m$.
26.             ave_dist = $sum\_global / N_m$ - 1
27.             for each $X_j$ ∈ Density-Object_List_Local do
28.                 distance = find_distance ($Q_m$, $X_j$)
29.                 if distance ≤ ave_dist then
30.                 add $X_j$ to $C_{im}$ (cluster $m^{th}$ of process i) and remove it from Density-Object_List_Local.
31.             end for
32.         end for
33.     end for
34.     if Density-Object_List_Local is not empty then
35.       for each $P_i$ , i ∈ {1, …, p} do
36.           $f_i$ = frequency_table_builder ($C_i$)    //$C_i$ = { $C_{i1}$,…,$C_{ik}$} all the clusters of $P_i$
37.           send $f_i$ to other processes and receive other processes' $f_i$.
38.           integrate all the $f_i$s to build the global frequency table F.
39.           Calculate real global modes M = {$m_1$,…, $m_k$}from global F.
40.           assign the remaining objects to the closest cluster based on their distance from the clusters'
              modes.
41.       end for
42.       each process $P_i$ sends its clusters $C_i$ = { $C_{i1}$,…,$C_{ik}$} to Master ($P_0$).
43.       Master integrate all these clusters and produce the final k initial clusters.
```

In line 5, *frequency_table_nc_builder* function builds a local frequency table of what a process has without considering any local clusters.

In line 8, for calculating the density of each object, please refer to Equations 1 and 2 of Section 3.1.

In line 19, for the function *find_distance*, we have used the hamming distance of part a in Section 2.2.

In line 40, to find the distance between the modes and the remaining objects, we have used the He's distance measure in Section 3.3.

Function 7: frequency_table_nc_builder

Input: d_i = each P_i's share of data, A = Set of attributes in the data, number of attributes: $m = |A|$

Output: frequency table of each process which is not clustered = f_nc

```
1. for each $A_j, j \in \{1, \ldots, m\}$
2.     for each $X_i \in d_i$
3.         find the count of all distinct values of $A_i$ and add it to $f\_nc$
4.     end for
5. end for
```

By plugging in the parallel implementation of RDM in PV3, we built parallel V3R (PV3R). This version works the same as PV3 but with non-random initialization seeds produced from parallel RDM.

In the next chapter, we will compare the performance of all the parallel versions (PV1, PV2, PV3, and PV3R) in terms of speed-up for the 15 UCI benchmark datasets we used in our experiments.

4.6. Parallel SIG RDM Initialization

As already mentioned in Section 4.4.5, SIG RDM initialization is evolved around calculating the distance between every unique value pair of each attribute in relation to every other attribute's distinct values. This process is highly time-consuming and therefore was a great opportunity to exploit our parallel resources. So, we parallelized Ahmad and Dey's algorithm [16] mentioned in Section 3.2.

One of our main challenges was to design a parallel algorithm that could work with the least amount of communication possible between the processes. Referring to line 6 of Algorithm 3 in Section 3.2, the main computation of this method is to calculate p $(w \mid x)$ and p $(\sim w \mid y)$, where (x, y) is a value pair of attribute A_i which we want to calculate their distance, and w is a subset of distinct values of other attributes. To do so, we proposed the idea of partial tables in which, instead of calculating the amount of these probabilities, the count of each relevant occurrence of these values in a process is stored. So, for example, for the pair (x, y) of attribute A_0, we formed a table with columns for each distinct value of attribute A_1, counted and stored the related occurrences in our partial table. We repeated the same process for all the other pairs and all the other distinct values of other attributes $(A_2, A_3, \ldots, A_m)$. In the communication phase, we exchanged these partial tables between the processes, traversed them, and summed up all the relevant occurrences to make one complete table. Using this table, we then computed the probabilities and calculated the unique distance between each unique pair of every attribute.

Algorithm 8: Parallel computation of the distance between each value pair of an attribute

Input: D = data objects, n = number of data objects, k = number of clusters, p = number of processes P_i, $i \in \{1, \ldots, p\}$

Output: *PD* = set of triplets (i, (x, y), Θ), where i is the number of the attribute, (x, y) is the pair of distinct values of A_i and Θ is their distance.

```
1.  for each $P_i$ , $i \in \{1, \ldots, p\}$ do
2.      read ($d_i$): $\frac{n}{p}$ distinct data objects from D.
3.  end for
4.  for each $P_i$ , $i \in \{1, \ldots, p\}$ do
5.      for each attribute $A_i$ do
6.          for each pair of categorical attribute values ($x$, $y$) do
7.              for every other attribute $A_j$ do
8.                  $partial\_table(i, x, y, w)$ = {# ($w$ | $x$), # (~$w$ | $y$), # $x$, # $y$)}
9.                  where $w$ is a subset of j^th attribute values and # is equivalent to count of.
10.             end for
11.         end for
12.     end for
13.     send my partial_table and receive others' partial_table to build the complete_table.
14.     for each attribute $A_i$ do
15.        for each pair of categorical attribute values ($x$, $y$) do
16.            Sum = 0
17.            for every other attribute $A_j$ do
17.                build $\theta(i, x, y)$ = max (p ($w$ | $x$) + p (~$w$ | $y$) – 1) from complete_table.
18.            end for
19.            $\Theta = \frac{Sum}{(m-1)}$, distance Θ between $x$ and $y$ values of attribute $A_i$
20.        end for
21.     end for
22. end for
```

Combined with parallel RDM initialization, we developed *Parallel SIG RDM Initialization* which reduced the processing time of distance calculation between each pair of an attribute. This resulted in an improvement of maximum 8.8 times speed-up with 9 processes for our largest dataset Nursery with 12,960 records.

We parallelized SIG-SCAV RDM using the parallel SIG RDM algorithm. The pseudo-code is similar to Algorithm 8 except for a small change in forming the initial clusters around the chosen centroids, as discussed in Algorithm 6 of Section 4.4.3.

4.7. Proposed Solution Framework (EPP)

In this section, we present a framework for categorical data clustering parallelization. Ensemble Parallelizing Process (EPP) framework uses all the different components which we developed and introduced in the previous sections in order to produce better clustering quality while enjoying parallelism.

Figure 4.4 shows a generic schematic of this framework:

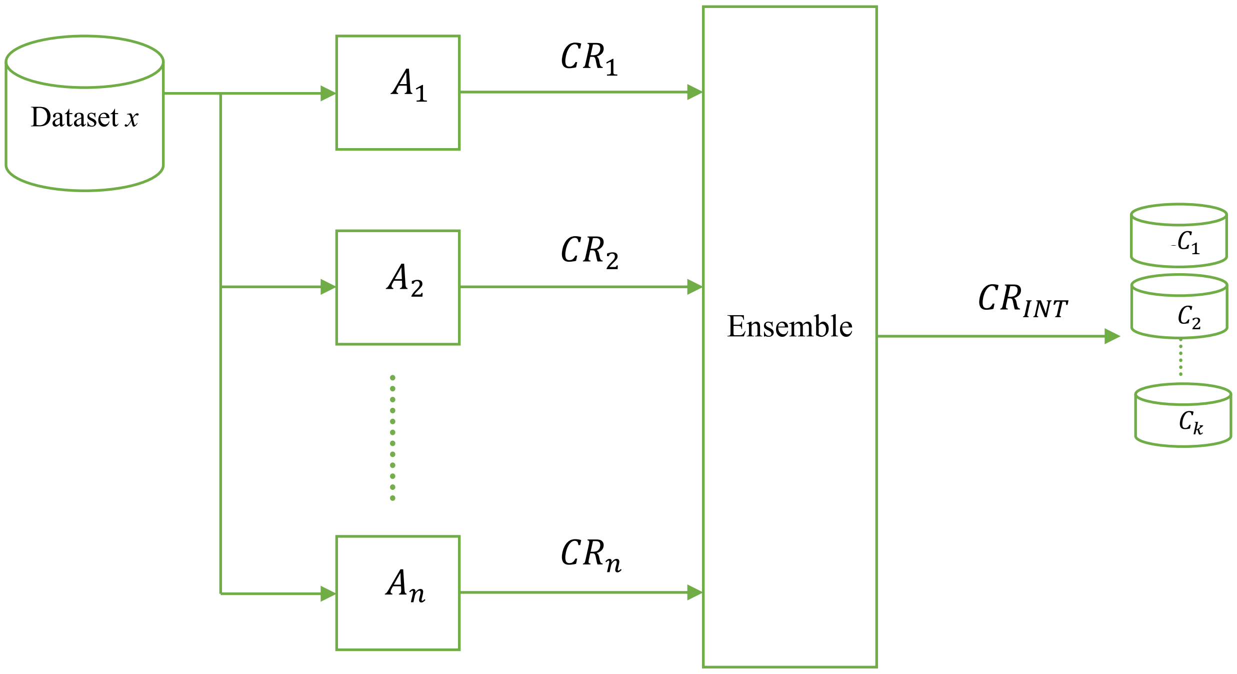

Figure 4.4 EPP Schematic

EPP receives a dataset x as the input and produces k clusters as the output. A unique index number is assigned to each data object to track it throughout the whole process. These indexed objects are clustered using n different clustering/initialization algorithms (A_1, A_2, …, A_n) producing n clustering results (CR_1, CR_2, …, CR_n). Each CR_i either contains the same or a different cluster label for each data object. To integrate all these labels for a single object, the ensemble process uses a majority-based approach by assigning the most frequent label to each data object. After obtaining the dominant cluster label for all the objects in the same way, EPP produces the final k clusters as the output (C_1, C_2,…, C_k). As previously mentioned, clustering categorical data is a subjective task as there is no universal dissimilarity measure for it. As a result, this consensus approach which takes into account the decision of the majority can lead into better overall clustering result.

Looking at Figure 4.4 of EPP, it is seen that we can parallelize this framework in two levels: vertically and horizontally. Our clustering/initialization algorithms can be run independently of each other. In other words, they are embarrassingly parallel (horizontal parallelization). On the other hand, each clustering/initialization algorithm A_i can be parallelized too (vertical parallelization).

We used our 6 different RDM initialization techniques mentioned from Section 4.4.1 to 4.4.6 as initialization algorithms, and then the PV3 which is a parallel k-modes based clustering algorithm (Section 4.3) was used to cluster these 6 different initial clusters produced by different versions of RDM. In the last step, we used the majority-based ensemble technique to produce the final clustering result which we refer to it as RDM ensemble, in the rest of this report. Figure 4.5 shows our realization of EPP:

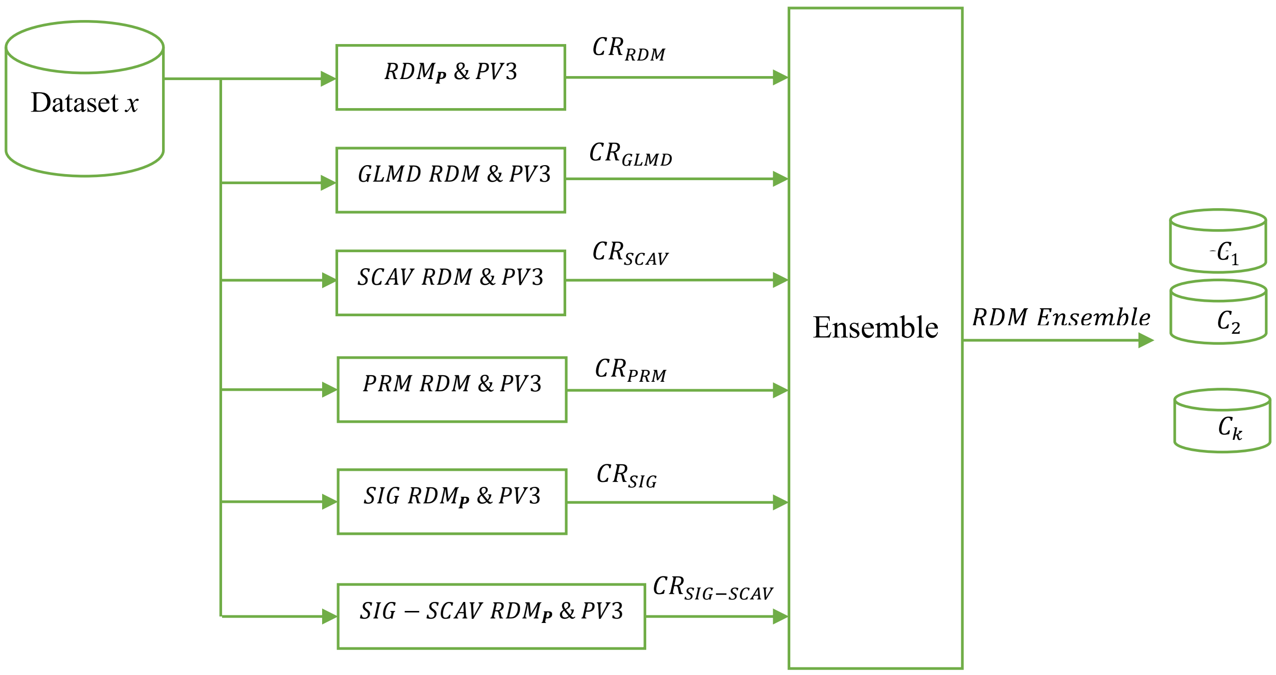

Figure 4.5 RDM Realization of EPP (RDM EPP)

The RDM ensemble's quality in terms of average accuracy (Table 5.9 in Section 5.3.2) are better than all the individual RDM initializations. It improves the initial RDM for 10 out of 15 datasets, remains similar in 3, and performs a bit worse in 2 datasets. RDM ensemble showed better clustering performance in terms of precision (Table 5.10) and recall (Table 5.11) comparing to each individual RDM too.

It is worth mentioning that among the 6 used RDM versions, 3 of them are fully parallelized: RDM (Section 4.5), SIG RDM, and SIG-SCAV (Section 4.6). We have not parallelized the GLBMD, SCAV, and PRM RDMs, and this is not a problem because the most intensive and time-consuming calculations belong to SIG and SIG-SCAV RDMs. As a result, the maximum elapsed time is one of their timings in an embarrassingly parallel architecture as in EPP.

EPP is a flexible framework, i.e., we can use it with different initialization/clustering algorithms, and with different levels of parallelism. For example, we can use the previously-produced initial clusters by different k-modes based sequential clustering algorithms, such as k-

PbC, or we can parallelize these algorithms to produce these initial clusters. Then, in both of these cases, we can use the PV3 algorithm as the main clustering algorithm as done in Figure 4.5, and obtain the *same* clustering result (CRs) as in the sequential algorithm. We can even use different clustering algorithms other than k-modes based algorithms, such as DBSCAN as our different A_is, and then perform the ensemble process to obtain the final clustering result, CR_{INT}. All these algorithms can either be parallelized or not.

As seen in Figure 4.5, the RDM EPP is a Multiple Program Multiple Data (MPMD) model. EPP is inherently a multiple program model, but what can make its RDM realization, a multiple data model is the fact that we are using different initialization algorithms (various RDM versions) to produce different initial clusters on the same dataset which can be considered as multiple data feeding into the EPP framework.

In Section 5.3.3, RDM ensemble's clustering quality is compared to other state-of-the-art clustering algorithms. We can observe that RDM ensemble has comparable clustering quality to other state-of-the-art k-modes clustering algorithms, and stands among the top three best, to the best of our knowledge.

4.8. Summary

In this chapter, we first explored parallelization of the k-modes clustering algorithm using Message Passing Interface (MPI). After exploring different ideas and techniques and choosing the best approach (PV3) in terms of both quality and speed-up, we tried to produce better clustering results which were deterministic by working on the seeding of the sequential algorithm. The sequential RDM initialization algorithm, and some of its variants were introduced for this purpose.

After improving and maintaining the clustering quality in the initialization phase of the k-modes algorithm, we parallelized these initialization techniques. First, we parallelized RDM algorithm which was a necessary first step to parallelize other RDM variants. The reason is that RDM variants are fairly similar to each other and only differ in small details such as calculating the distance between the data objects. We then parallelized SIG RDM and SIG-SCAV RDM algorithms which helped us achieve good speed-ups in the initialization phase of the k-modes algorithm.

In the penultimate section of this chapter, we presented an Ensemble Parallelizing Process (EPP) framework for parallelizing categorical data clustering. EPP is a flexible framework which can produce better clustering quality while providing different levels of parallelism. We then plugged in all the techniques and algorithms developed and mentioned in this chapter to build an RDM realization of EPP (RDM EPP). The clustering quality of the output (RDM ensemble) is among the top three best k-modes based clustering algorithms to the best of our knowledge.

Chapter 5. Experiments and Results

In this chapter, we present the datasets utilized for evaluating the proposed parallelization solutions for clustering categorical data. This is followed by the explanation of the carried-out experiments to evaluate the performance and the results.

5.1. Datasets for Experimental Evaluations

We used 15 bench mark categorical datasets with different characteristics from the UCI Machine Learning Repository [22] in our performance evaluation experiments. Table 5.1 provides information about these datasets.

#	Dataset	# Clusters	# Instances	# Attributes
1	Balance scale	3	625	4
2	Breast cancer	2	699	9
3	Car evaluation	4	1,728	6
4	Chess	2	3,196	36
5	Dermatology	6	366	34
6	Lung cancer	3	32	56
7	Lymphography	4	148	18
8	Mushroom	2	8,124	22
9	Nursery	5	12,960	8
10	Solar flare	3	1,066	12
11	Soybean	4	47	35
12	Splice	3	3,190	60
13	Tic-tac-toe	2	958	9
14	Vote	2	435	16
15	Zoo	7	101	17

Table 5.1 Experimental Datasets

5.2. Parallel K-Modes (PV1, PV2, PV3, and PV3R)

In this section, we measure and report the performance of different parallelization methods developed, in terms of speed-up. We have obtained these figures by running our programs on a Linux based Beowulf cluster named Apini available at Concordia University.

The quality has been excluded from these comparisons because the algorithms PV1, PV2, and PV3 use the original k-modes algorithm with random initialization. Conducting a quality-wise comparison between them would be irrelevant since each algorithm starts from distinct random initial clusters in every run, which are not equivalent to those of the other algorithms. On the other hand, the PV3R algorithm demonstrates an identical clustering quality to that of the RDM algorithm, due to their shared initialization technique. The clustering results of the RDM algorithm, including accuracy, precision, and recall can be found in Tables 5.9, 5.10, and 5.11 within Section 5.3.2.

5.2.1. Speed-up

Since speed-up is meaningful in the number of processes (*p*) used, it would be quite meaningless to calculate the average speed-up produced on different number of processes. As a result, for the purpose of comparison, we have used the maximum speed-up for each version.

For each dataset, we ran algorithms PV2, PV3 and PV3R 100 times with different number of processes ranging from 2 to 80 (maximum available processes) to measure the average elapsed time for every one of these algorithms. We report the maximum speed-up obtained in Table 5.2. For PV1, since we only used a certain number of processes, equal to the number of clusters *k*, we ran the experiment 100 times with *k* processes and reported only one speed-up.

#Instances	Speed-up (MAX)	PV1	PV2	PV3	PV3R
32	*Lung cancer*	**2.67**	1.00	0.76	0.71
47	*Soybean*	**2.83**	1.31	1.06	1.38
96	*Zoo*	**4.10**	1.46	1.11	1.47
148	*Lymphography*	**3.60**	1.71	1.33	1.32
366	*Dermatology*	1.10	2.94	**3.06**	2.47
434	*Vote*	1.15	**3.33**	**3.33**	2.37
624	*Balance scale*	3.50	3.50	**4.67**	2.22
698	*Breast cancer*	1.29	**5.80**	2.42	2.64
958	*Tic-tac-toe*	0.81	1.62	2.10	**4.45**
1,066	*Solar flare*	2.46	**5.21**	3.44	4.15
1,728	*Car evaluation*	3.15	8.73	**12.00**	3.70
3,190	*Splice*	1.03	5.66	**6.44**	4.84
3,196	*Chess*	0.52	6.35	**11.27**	7.71
8,192	*Mushroom*	0.15	8.13	**10.54**	9.68
12,960	*Nursery*	0.52	**12.13**	10.10	9.22

Table 5.2 Speed-up

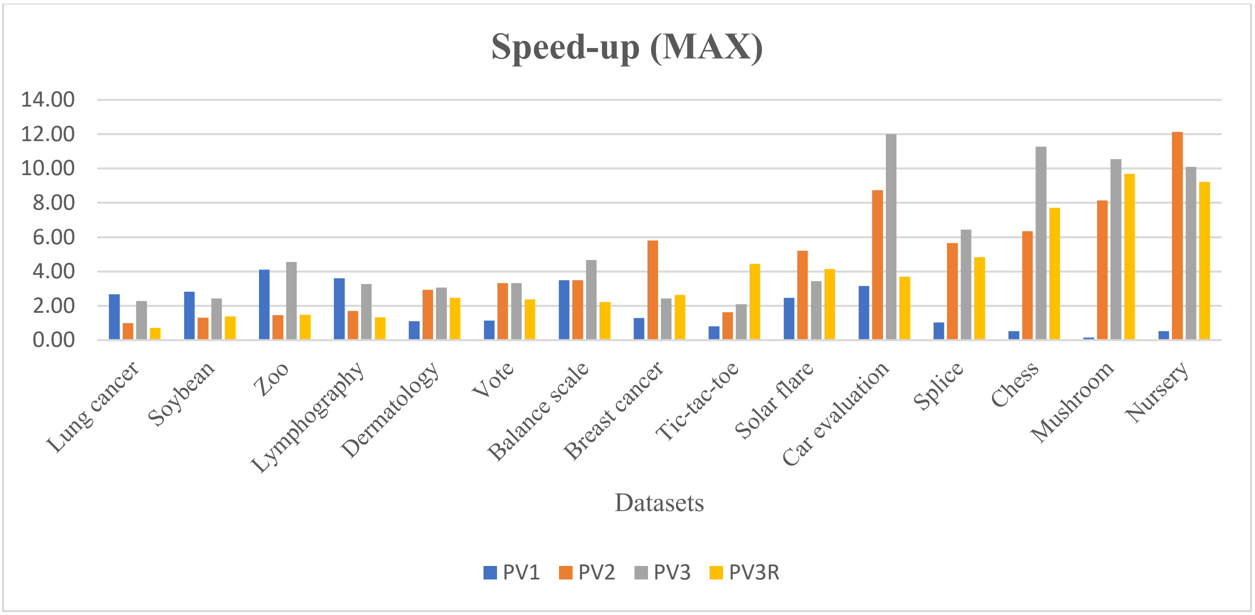

Figure 5.1 Maximum Speed-up Comparison Between PV1, PV2, PV3, and PV3R

a) Relatively Small Datasets

As per Figure 5.1, In small datasets, PV1 has almost the best speed-up among all other versions. In comparison with PV3R, it has been expected because in PV3R, more work (the initialization process) is done and the extra effort to form better initial clusters in parallel is simply not worth it. Also in small datasets, PV2 cannot benefit much by using an increased number of processes due to the large overhead of data exchange incurred at every iteration.

PV3 and PV1, on the other hand, have an almost equal speed-up due to their nearly identical approach. We should also mention that when the size of the input dataset is small, using fewer processes works better, in general, in terms of speed-up. This is because in this case, by increasing the number of processes, we are unwillingly adding on the communication cost while each process does less computation, and this can negatively affect the speed-up and efficiency.

b) Relatively Large Datasets

As seen in Figure 5.1, PV3 is the fastest in working with relatively large datasets while we expected PV2 to have the best speed-up among all the versions. When we examine the results more carefully, we notice that the number of iterations in PV3 is less than that of PV2. Again, since PV2 suffers from information loss along the way, in particular for large datasets when using more processes, this can lead to doing more work, and it increases the number of iterations for k-modes.

PV1 is very slow as expected because it is bound to using only a certain number of processes, and that turns into a major issue in working with large datasets.

PV3R has an acceptable speed-up in comparison to any other PVs. While it is not as high as PV3 and PV2 because of the time it takes to do the initialization, PV3 starts off from better initial clusters which helps it to converge faster, and that in turn makes up for the total elapsed time.

c) Speed-up vs. Number of Processes (*p*)

To have a general picture of the relation between the size of the dataset, number of processes, and speed-up, we picked seven UCI datasets with sizes from small to large and measured the maximum speed-up for PV2, PV3, and PV3R. PV1 has been excluded from this experiment because of its limitation of using different number of processes.

#Instances	**#Processe** / **Datasets**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**12**	**14**	**16**	**32**	**64**	**80**
148	*Lymphography*	1.13	1.44	1.64	1.64	1.64	**1.71**	**1.71**	1.20	1.20	1.20	1.16	1.16	0.80	0.72	0.55
434	*Vote*	1.25	2.14	2.50	2.50	2.73	**3.33**	**3.33**	2.73	2.73	2.73	2.73	2.73	1.58	0.79	0.60
698	*Breast cancer*	1.66	2.90	3.41	4.14	4.83	5.27	**5.80**	5.27	5.27	5.27	4.46	5.27	2.90	1.53	1.07
1,728	*Car evaluation*	2.00	3.20	5.65	4.57	4.92	5.82	7.38	5.19	5.65	6.19	5.65	**8.73**	6.62	0.91	0.60
3,196	*Chess*	1.31	1.27	1.41	1.96	2.55	3.08	3.54	3.03	3.65	3.62	4.73	5.44	**6.35**	4.91	3.14
8,192	*Mushroom*	1.53	2.32	1.67	4.19	3.46	2.99	2.85	4.25	2.64	3.29	3.58	4.96	7.99	**8.13**	4.74
12,960	*Nursery*	1.89	3.25	4.20	5.65	6.41	6.92	7.65	7.14	7.00	6.83	8.83	10.17	**12.13**	3.89	3.49

Table 5.3 PV2's Speed-up vs. #Processes

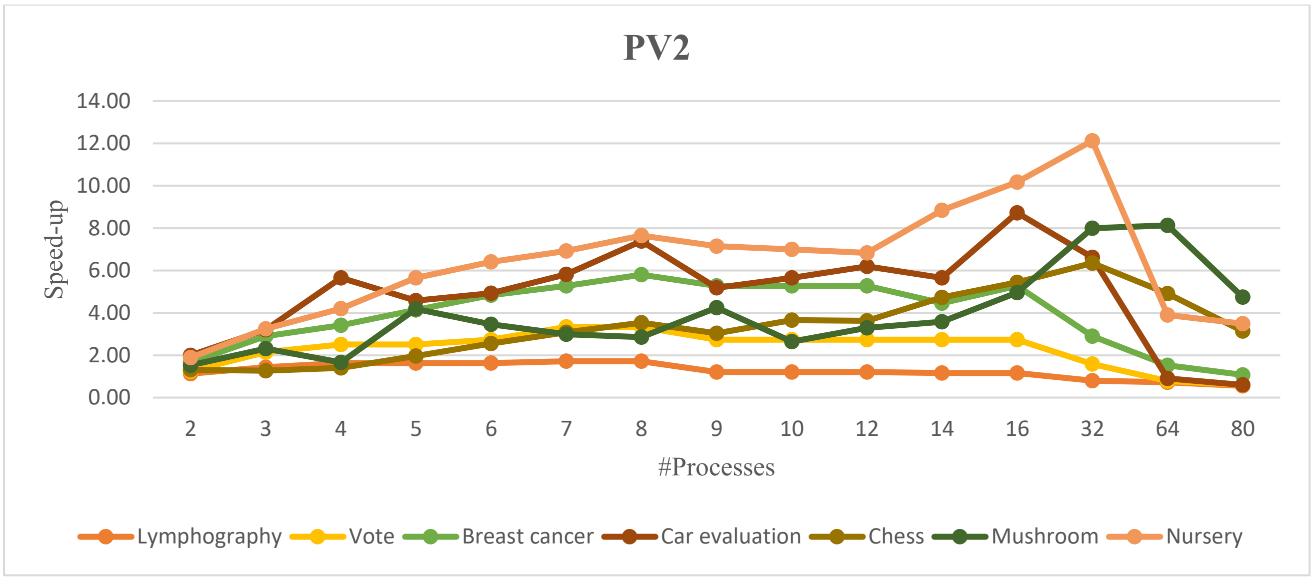

Figure 5.2 PV2's Speed-up

#Instances	#Processe / Datasets	2	3	4	5	6	7	8	9	10	12	14	16	32	64	80
148	*Lymphography*	0.92	1.16	**1.24**	1.33	1.16	**1.24**	1.20	0.97	0.80	0.75	0.72	0.75	0.49	0.14	0.13
434	*Vote*	1.15	1.88	2.14	2.73	2.73	**3.33**	3.00	2.31	2.31	1.31	2.00	1.76	0.97	0.17	0.15
698	*Breast cancer*	1.38	1.93	2.15	**2.42**	2.23	2.15	2.15	1.76	1.66	1.49	1.53	1.38	0.53	0.20	0.18
1,728	*Car evaluation*	1.70	3.00	8.00	4.17	4.68	5.05	9.14	4.47	4.68	6.62	4.36	**12.00**	7.38	1.59	0.51
3,196	*Chess*	1.35	2.50	6.84	4.47	3.60	5.06	6.72	**11.27**	6.72	4.71	4.68	6.96	5.44	4.85	1.40
8,192	*Mushroom*	1.09	3.48	2.52	6.16	3.58	7.26	6.87	5.29	4.87	6.78	5.37	**10.54**	7.78	2.22	1.67
12,960	*Nursery*	2.11	3.07	4.74	5.66	6.05	8.47	8.05	6.78	8.01	8.09	**10.10**	9.48	8.81	2.91	2.53

Table 5.4 PV3’s Speed-up vs. #Processes

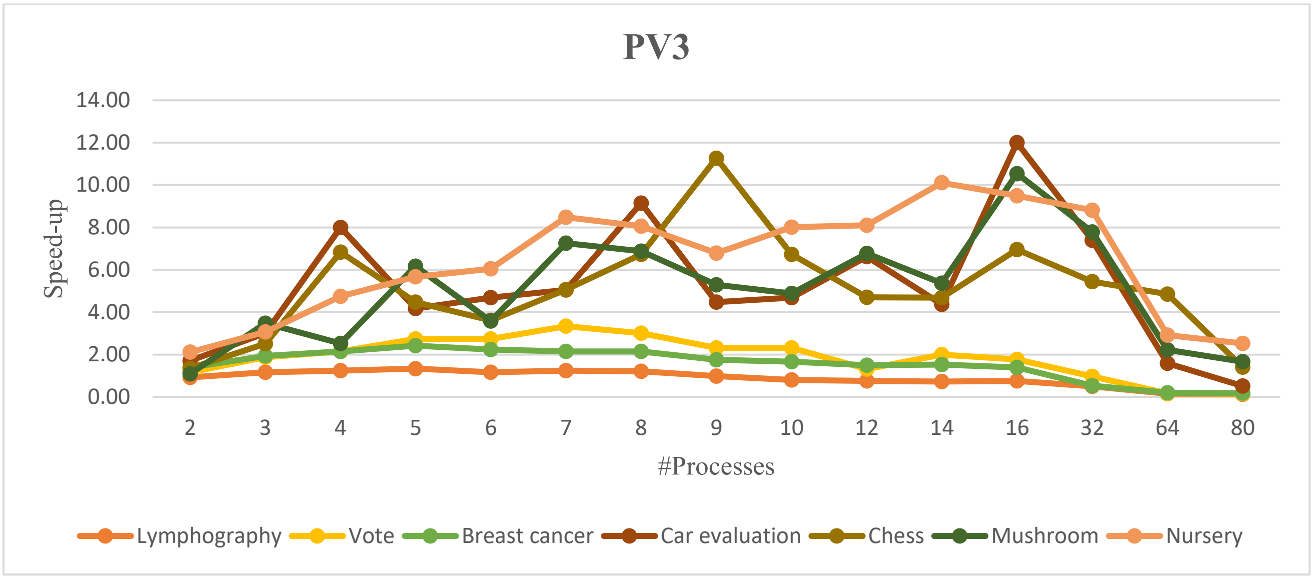

Figure 5.3 PV3’s Speed-up

#Instances	#Processe / Datasets	2	3	4	5	6	7	8	9	10	12	14	16	32	64	80
148	*Lymphography*	0.91	1.21	**1.32**	1.16	1.12	1.12	0.97	0.91	0.85	0.78	0.74	0.74	0.45	0.07	0.06
434	*Vote*	1.32	2.14	**2.65**	2.05	2.14	1.96	2.37	1.88	1.80	1.67	1.61	0.83	0.40	0.11	0.09
698	*Breast cancer*	1.27	2.00	2.28	2.44	**2.64**	**2.64**	**2.64**	2.13	2.06	1.94	1.83	1.83	1.10	0.21	0.16
1,728	*Car evaluation*	1.36	2.50	2.53	3.08	3.19	3.49	**3.70**	3.25	3.03	2.94	3.14	3.14	1.85	0.30	0.23
3,196	*Chess*	1.45	2.61	3.43	4.18	4.98	5.72	6.32	5.47	5.60	6.08	6.81	**7.71**	6.67	1.84	1.42
8,192	*Mushroom*	1.44	2.60	3.44	4.32	5.06	5.72	6.56	5.97	6.21	6.48	7.71	9.10	**9.68**	2.84	2.24
12,960	*Nursery*	1.33	2.79	3.40	4.22	5.40	5.74	6.49	6.07	5.40	6.32	6.66	**9.22**	7.51	1.67	1.35

Table 5.5 PV3R’s Speed-up vs. #Processes

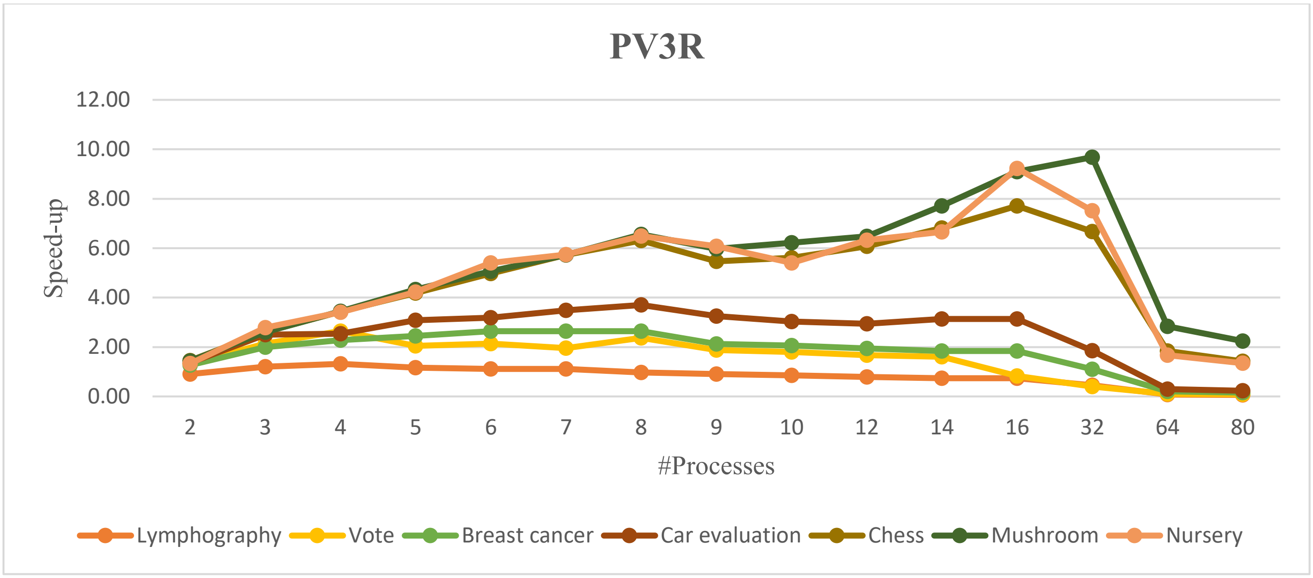

Figure 5.4 PV3R's Speed-up

After examining Tables 5.3, 5.4, 5.5, and their corresponding charts, we noted that in all our parallel versions, the maximum speed-up happens in smaller number of processes when working with smaller datasets while for larger datasets, we have the highest speed-ups when using higher number of processes. This is reasonable because when working with small datasets, using a large number of processes leads to more communication overhead while there is not much computation to outweigh overhead and vice versa.

Another point which stands out by looking at and comparing the data and the charts is that the curves for PV3R are almost smooth and predictable while it is not the case for PV2 and PV3. The reason lies in the initialization phase of PV3R. As already mentioned in chapter 4, with non-random initialization of k-modes, we can produce deterministic and better clustering results which do not change and are not order-dependent. As a result, the amount of work (the number of iterations) does not change in different runs, hence explaining why there are not as many peaks and valleys in PV3R's corresponding chart.

On the other hand, PV2 and PV3 use random initialization, and that leads to lack of predictability of the number of the iterations for k-modes to converge. This, in turn, affects the elapsed time and consequently speed-up.

5.3. RDM

As mentioned before in Section 4.4.1, RDM is an adopted version of Cao's algorithm which prioritizes density over distance in order to produce better initial clusters. In this section, we first compare the performance of RDM and its parent algorithm Cao. Then, we compare RDM with the results of the other RDM variants. Next, we compare the quality of RDM ensemble to other state-of-the-art k-modes based algorithms for clustering. These results include accuracy, precision, and recall which were obtained from running 10 different algorithms on 15 UCI datasets. Dinh and Huynh [7] collected all these results which we used for the purpose of comparison too.

The winner in all these comparisons is k-PbC algorithm [7] which as mentioned in Section 3.4 uses an FP-tree (Frequent Pattern tree) to mine frequent patterns of attribute values to form initial clusters.

It is worth mentioning that even though k-PbC yields better quality than other algorithms in terms of accuracy, precision, and recall, its clustering quality is not significantly different, if not comparable to, those of the other algorithms. For example, when comparing the accuracy of k-PbC and the next best algorithm (CD-Clustering algorithm apart from RDM ensemble or RDM), in 8 datasets, their difference is in the hundredth decimal places, and in 2 datasets, it is in the thousandth decimal places. Only in 4 datasets, this difference is in one decimal place (10%).

5.3.1. RDM vs. Cao

To compare the quality of RDM and Cao, we used accuracy, precision, and recall in our experiments. As can be seen in Table 5.6, RDM is better in accuracy for 8 datasets and is the same to Cao's algorithm in 1 dataset:

Datasets	**Cao**	**RDM**
Balance scale	0.3760	**0.4454**
Breast cancer	**0.9113**	0.8653
Car evaluation	**0.4936**	0.3403
Lung cancer	0.5313	**0.5938**
Chess	0.5663	**0.5673**
Dermatology	**0.5874**	0.5109
Lymphography	0.3514	**0.5135**
Mushroom	0.8754	**0.8902**
Nursery	**0.3673**	0.3147
Solar flare	**0.5432**	0.4747
Soybean-small	**1.0000**	**1.0000**
Splice	**0.4009**	0.3282
Tic-tac-toe	0.6106	**0.6837**
Vote	0.8644	**0.8714**
Zoo	0.6733	**0.9583**

Table 5.6 Accuracy of RDM vs. Cao

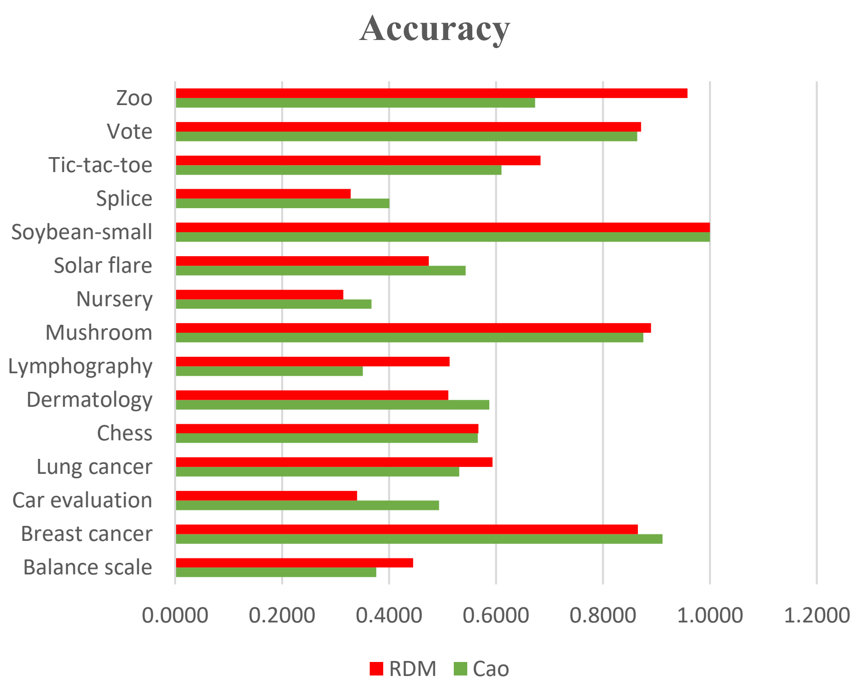

Figure 5.5 Accuracy of RDM vs. Cao

Datasets	**Cao**	**RDM**
Balance scale	0.3282	**0.4072**
Breast cancer	**0.9292**	0.9068
Car evaluation	**0.3826**	0.3235
Lung cancer	0.5468	**0.6435**
Chess	**0.5796**	0.5699
Dermatology	**0.5604**	0.3877
Lymphography	0.2698	**0.4281**
Mushroom	0.9019	**0.9056**
Nursery	0.2978	**0.3062**
Solar flare	0.3381	**0.3384**
Soybean-small	**1.0000**	**1.0000**
Splice	**0.4518**	0.3011
Tic-tac-toe	0.5859	**0.6635**
Vote	0.8568	**0.8668**
Zoo	0.5996	**0.9472**

Table 5.7 Precision of RDM vs. Cao

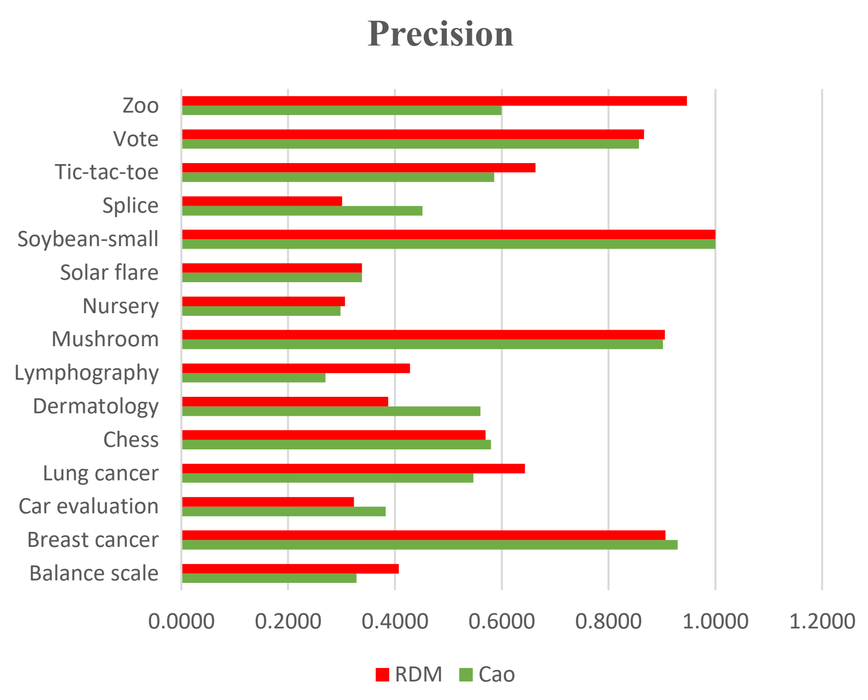

Figure 5.6 Precision of RDM vs. Cao

Datasets	**Cao**	**RDM**
Balance scale	0.3228	**0.3920**
Breast cancer	**0.8773**	0.8071
Car evaluation	**0.4875**	0.3257
Lung cancer	0.5507	**0.5675**
Chess	0.5537	**0.5583**
Dermatology	**0.5260**	0.3889
Lymphography	0.2955	**0.6971**
Mushroom	0.8709	**0.8868**
Nursery	0.2273	**0.2495**
Solar flare	**0.4310**	0.4080
Soybean-small	**1.0000**	**1.0000**
Splice	**0.4379**	0.2971
Tic-tac-toe	0.5917	**0.6759**
Vote	0.8730	**0.8865**
Zoo	0.6233	**0.9544**

Table 5.8 Recall of RDM vs. Cao

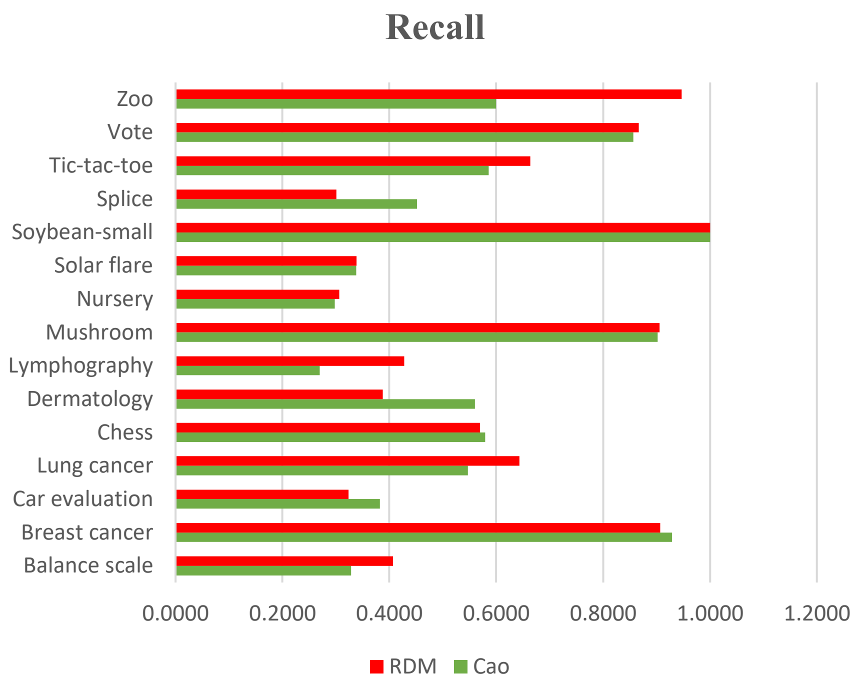

Figure 5.7 Recall of RDM vs. Cao

RDM's precision and recall are better in 9 out of 15 datasets and are the same as Cao's in 1 dataset according to Tables 5.7 and 5.8.

5.3.2. RDM, its Variants and RDM Ensemble

In this section, the accuracy, precision and recall of RDM along with its 5 other variants and finally RDM ensemble (RDM EPP's output) are presented:

Datasets	**RDM [9]**	**GLMD RDM [40]**	**SCAV RDM**	**PRM RDM [15]**	**SIG RDM [16]**	**SIG-SCAV RDM [16]**	**RDM Ensemble**
Balance scale	0.4454	0.4467	0.4125	0.4443	0.4683	0.4799	**0.5224**
Breast cancer	0.8653	0.8653	0.8653	0.8653	**0.9728**	0.7593	0.8653
Car evaluation	0.3403	0.3441	0.3332	0.3379	0.3421	0.3534	**0.3652**
Lung cancer	0.5938	0.5313	0.5313	0.5938	0.4688	0.4688	**0.6875**
Chess	0.5673	0.5760	**0.5917**	0.5673	0.5200	0.5760	0.5760
Dermatology	0.5109	0.4699	**0.7760**	0.5082	0.6421	0.6421	0.6011
Lymphography	0.5135	0.5068	0.5000	0.4527	0.4662	0.4054	**0.6081**
Mushroom	0.8902	0.8902	0.8902	0.8902	**0.8983**	**0.8983**	0.8902
Nursery	0.3147	0.3249	0.3085	0.3127	0.3371	0.3134	**0.3613**
Solar flare	0.4747	0.4747	0.4578	0.4803	0.4625	0.4053	**0.4925**
Soybean-small	**1.0000**	0.7021	0.8445	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Splice	0.3282	0.3580	0.4154	0.3254	**0.4749**	0.4370	0.4169
Tic-tac-toe	**0.6837**	0.6837	0.6701	**0.6837**	0.6159	0.6207	0.6263
Vote	0.8714	0.8719	0.8664	0.8719	0.8733	**0.8756**	0.8733
Zoo	**0.9583**	0.8542	0.7708	**0.9583**	0.7132	0.6725	0.9063
Average Accuracy	0.624	0.593	0.616	0.619	0.617	0.594	**0.653**

Table 5.9 Accuracy of RDM's Variants and RDM Ensemble

Datasets	**RDM [9]**	**GLMD RDM [40]**	**SCAV RDM**	**PRM RDM [15]**	**SIG RDM [16]**	**SIG-SCAV RDM [16]**	**RDM Ensemble**
Balance scale	0.4072	0.3775	0.3982	0.4102	0.4409	0.4436	**0.4624**
Breast cancer	0.9068	0.9068	0.9068	0.9068	**0.9663**	0.7790	0.9068
Car evaluation	0.3235	0.3220	0.3294	0.3219	0.3261	0.3176	**0.3381**
Lung cancer	0.6435	0.6852	0.5278	0.6435	0.5261	0.4471	**0.7269**
Chess	0.5699	0.5699	**0.5945**	0.5699	0.5125	0.5811	0.5799
Dermatology	0.3877	0.3880	**0.7055**	0.3880	0.5620	0.5889	0.3880
Lymphography	0.4281	**0.4530**	0.4494	0.4175	0.3224	0.3562	0.4423
Mushroom	0.9056	0.9056	0.9056	0.9056	**0.9170**	**0.9170**	0.9056
Nursery	0.3062	0.3176	0.3049	0.3042	0.3256	0.3052	**0.3589**
Solar flare	0.3384	0.3385	0.3390	0.3393	0.3387	0.3295	**0.3397**
Soybean-small	**1.0000**	**1.0000**	0.8495	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Splice	0.3011	0.2587	0.4258	0.2979	**0.5217**	0.4313	0.3855
Tic-tac-toe	**0.6635**	0.5365	0.6556	**0.6635**	0.6149	0.6095	0.6123
Vote	0.8668	0.8673	0.8626	0.8672	0.8677	**0.8697**	0.8684
Zoo	**0.9472**	0.7465	0.6035	**0.9472**	0.5548	0.5023	0.7465
Average Precision	0.600	0.578	0.591	0.599	0.586	0.565	**0.604**

Table 5.10 Precision of RDM's Variants and RDM Ensemble

Datasets	RDM [9]	GLMD RDM [40]	SCAV RDM	PRM RDM [15]	SIG RDM [16]	SIG-SCAV RDM [16]	RDM Ensemble
Balance scale	0.3920	0.3715	0.3837	0.3944	0.3824	0.3913	**0.4005**
Breast cancer	0.8071	0.8071	0.8071	0.8071	**0.9743**	0.8057	0.8071
Car evaluation	0.3257	0.2987	0.3304	0.3235	0.3221	0.3181	**0.3498**
Lung cancer	0.5675	0.6046	0.5581	0.5675	0.4684	0.5031	**0.6749**
Chess	0.5583	0.5583	**0.5846**	0.5583	0.5099	0.5668	0.5673
Dermatology	0.3889	0.4059	**0.7037**	0.3874	0.5989	0.5778	0.4620
Lymphography	**0.6971**	0.6398	0.6828	0.4194	0.3662	0.3918	0.6204
Mushroom	0.8868	0.8868	0.8868	0.8868	**0.8946**	**0.8946**	0.8868
Nursery	0.2495	0.2860	0.2547	0.2559	0.2912	0.2782	**0.3117**
Solar flare	0.4080	0.4159	0.4854	0.4929	0.4870	0.1357	**0.4970**
Soybean-small	**1.0000**	**1.0000**	0.8527	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Splice	0.2971	0.2813	0.4415	0.2941	**0.4929**	0.4533	0.3815
Tic-tac-toe	**0.6759**	0.5401	0.6698	**0.6759**	0.6269	0.6200	0.6228
Vote	0.8865	0.8870	0.8824	0.8869	0.8869	**0.8888**	0.8880
Zoo	**0.9544**	0.8115	0.6479	**0.9544**	0.5745	0.5454	0.8115
Average Recall	0.606	0.586	0.611	0.594	0.592	0.558	**0.619**

Table 5.11 Recall of RDM's Variants and RDM Ensemble

As seen, the quality of RDM ensemble is better than individual RDMs, in general. It has a higher average accuracy, precision, and recall for all the benchmark datasets we used in our experiments. In addition, it has a higher number of best accuracies, precisions, and recalls in all the datasets compared to individual RDM algorithms.

5.3.3. RDM Ensemble vs. Other Algorithms

In this section, RDM ensemble's quality is compared with the top 2 best state-of-the-art k-modes based clustering algorithms, namely CD-Clustering and k-PbC. The inclusion of Cao's algorithm in our comparisons is attributed to its role as our primary algorithm.

A full comparison between RDM ensemble and 10 state-of-art k-modes based clustering algorithms is presented in Appendix A, Tables A.1 to A.3.

Datasets	**Cao**	**CD-Clustering**	**k-PbC**	**RDM ensemble**
Balance scale	0.3760	0.4129	**0.5680**	0.5224
Breast cancer	0.9113	0.9514	**0.9614**	0.8653
Car evaluation	**0.4936**	0.3125	0.3640	0.3652
Lung cancer	0.5313	0.5938	**0.6875**	**0.6875**
Chess	0.5663	0.5156	0.5047	**0.5760**
Dermatology	0.5874	0.8552	**0.9645**	0.6011
Lymphography	0.3514	0.5000	0.5946	**0.6081**
Mushroom	0.8754	0.7244	0.8861	**0.8902**
Nursery	0.3673	0.4156	**0.4492**	0.3613
Solar flare	0.5432	0.4343	**0.5901**	0.4925
Soybean-small	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Splice	0.4009	0.7260	**0.7746**	0.4169
Tic-tac-toe	0.6106	0.6044	**0.6326**	0.6263
Vote	0.8644	0.8713	**0.8805**	0.8733
Zoo	0.6733	0.8218	0.8911	**0.9063**
Average Accuracy	0.610	**0.649**	**0.717**	**0.653**

Table 5.12 Accuracy of Compared Algorithms

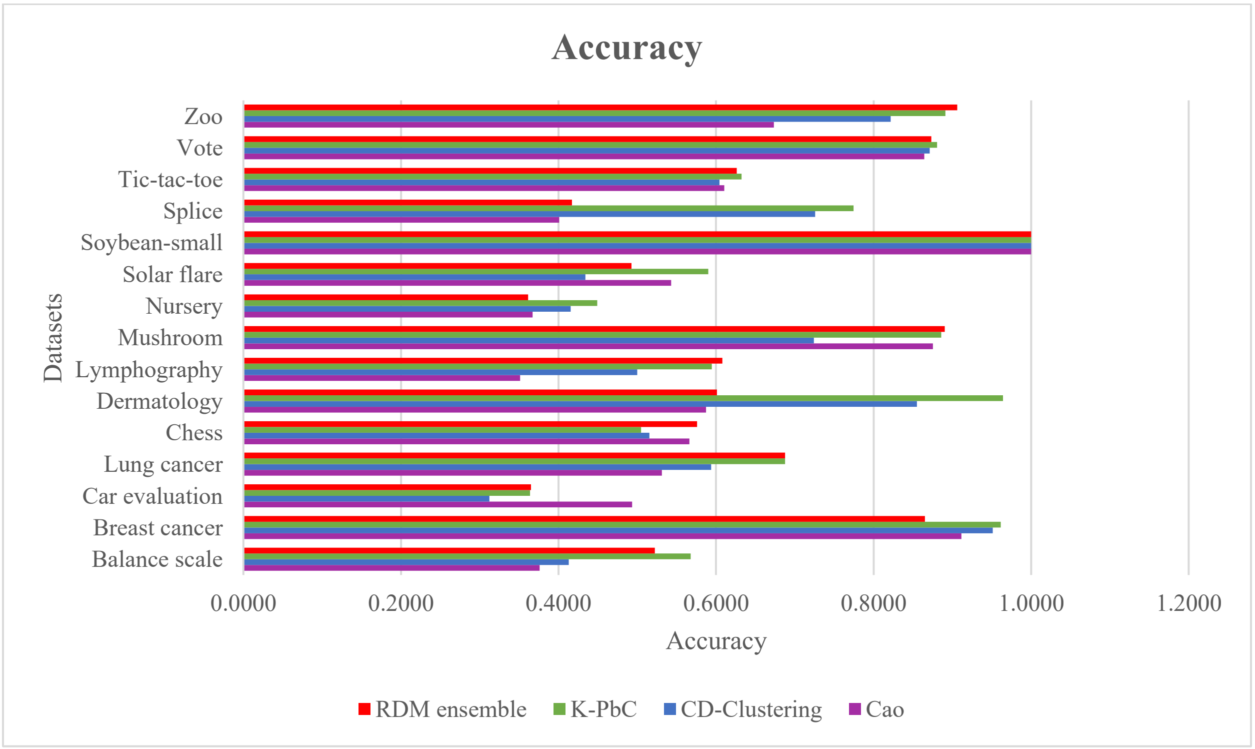

Figure 5.8 Accuracy of Compared Algorithms

As can be seen in Table 5.12 and Figure 5.8, after k-PbC, RDM ensemble has a higher number of best accuracies. K-PbC has the best accuracies in 10 (out of 15) datasets while RDM ensemble has the best accuracies in 6 datasets.

Regarding the average accuracy, k-PbC is the best while RDM ensemble and CD-clustering algorithms stand in the second and third places, respectively.

Datasets	**Cao**	**CD-Clustering**	**k-PbC**	**RDM ensemble**
Balance scale	0.3282	0.3751	**0.4825**	0.4624
Breast cancer	0.9292	0.9470	**0.9517**	0.9068
Car evaluation	**0.3826**	0.3125	0.3704	0.3381
Lung cancer	0.5468	0.5980	**0.7421**	0.7269
Chess	0.5796	0.5162	0.5095	**0.5799**
Dermatology	0.5604	0.7543	**0.9612**	0.3880
Lymphography	0.2698	0.4724	**0.4752**	0.4423
Mushroom	0.9019	0.7990	0.9000	**0.9056**
Nursery	0.2978	**0.6494**	0.4352	0.3589
Solar flare	0.3381	0.3377	**0.3425**	0.3397
Soybean-small	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Splice	0.4518	0.7198	**0.7815**	0.3855
Tic-tac-toe	0.5859	0.5746	**0.6396**	0.6123
Vote	0.8568	0.8670	**0.8762**	0.8684
Zoo	0.5996	0.5688	**0.7503**	0.7465
Average Precision	0.575	**0.633**	**0.681**	**0.604**

Table 5.13 Precision of Compared Algorithms

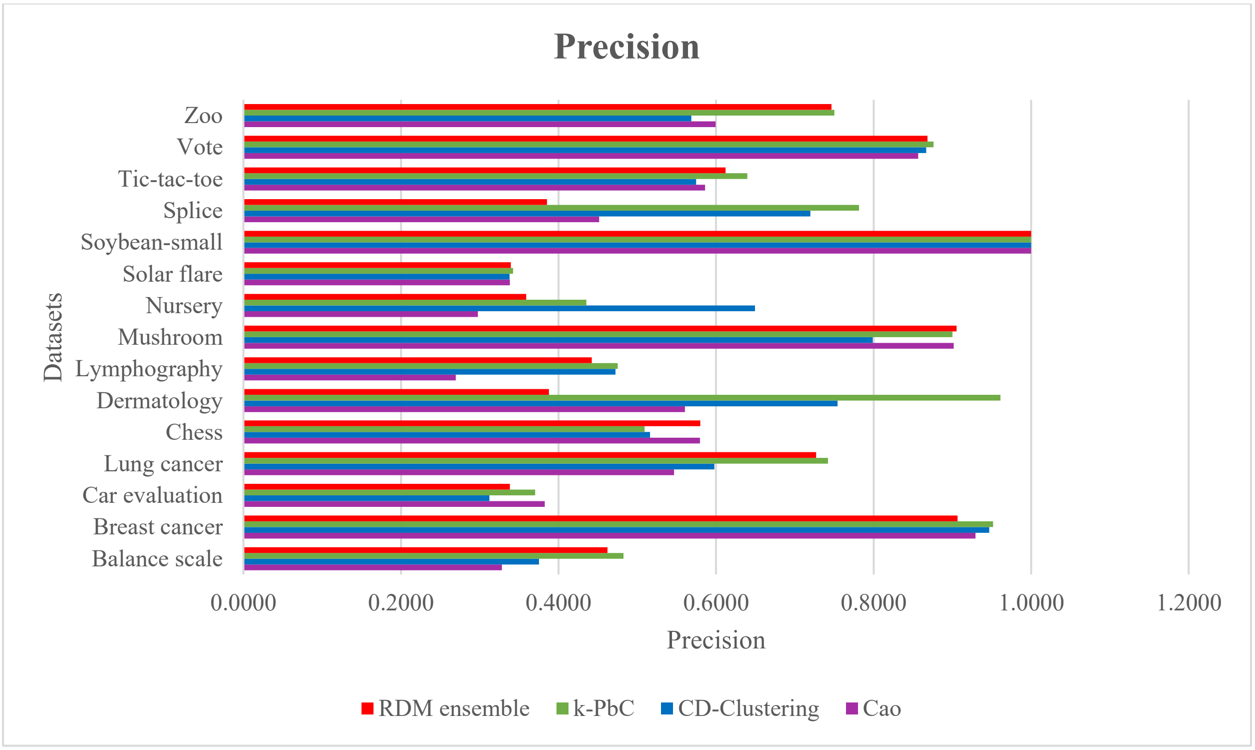

Figure 5.9 Precision of Compared Algorithms

As seen in Table 5.13 and Figure 5.9, k-PbC has the highest number of best precisions (11 out of 15 datasets) and after that, RDM ensemble algorithm stands in the second place with 3 best precisions.

As for the average precision, k-PbC, CD-clustering, and RDM ensemble are the 3 best algorithms, respectively.

Datasets	**Cao**	**CD-Clustering**	**k-PbC**	**RDM ensemble**
Balance scale	0.3228	0.3540	**0.4673**	0.4005
Breast cancer	0.8773	0.9550	**0.9656**	0.8071
Car evaluation	**0.4875**	0.4114	0.4465	0.3498
Lung cancer	0.5507	0.6208	**0.6900**	0.6749
Chess	0.5537	0.5162	0.5091	**0.5673**
Dermatology	0.5260	0.8189	**0.9679**	0.4620
Lymphography	0.2955	**0.6255**	0.5579	0.6204
Mushroom	0.8709	0.7151	0.8829	**0.8868**
Nursery	0.2273	0.2569	**0.3370**	0.3117
Solar flare	0.4310	0.4775	0.4467	**0.4970**
Soybean-small	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Splice	0.4379	0.7722	**0.8421**	0.3815
Tic-tac-toe	0.5917	0.5785	**0.6538**	0.6228
Vote	0.8730	0.8863	**0.8960**	0.8880
Zoo	0.6233	0.6860	0.7860	**0.8115**
Average Recall	0.578	**0.645**	**0.697**	**0.619**

Table 5.14 Recall of Compared Algorithms

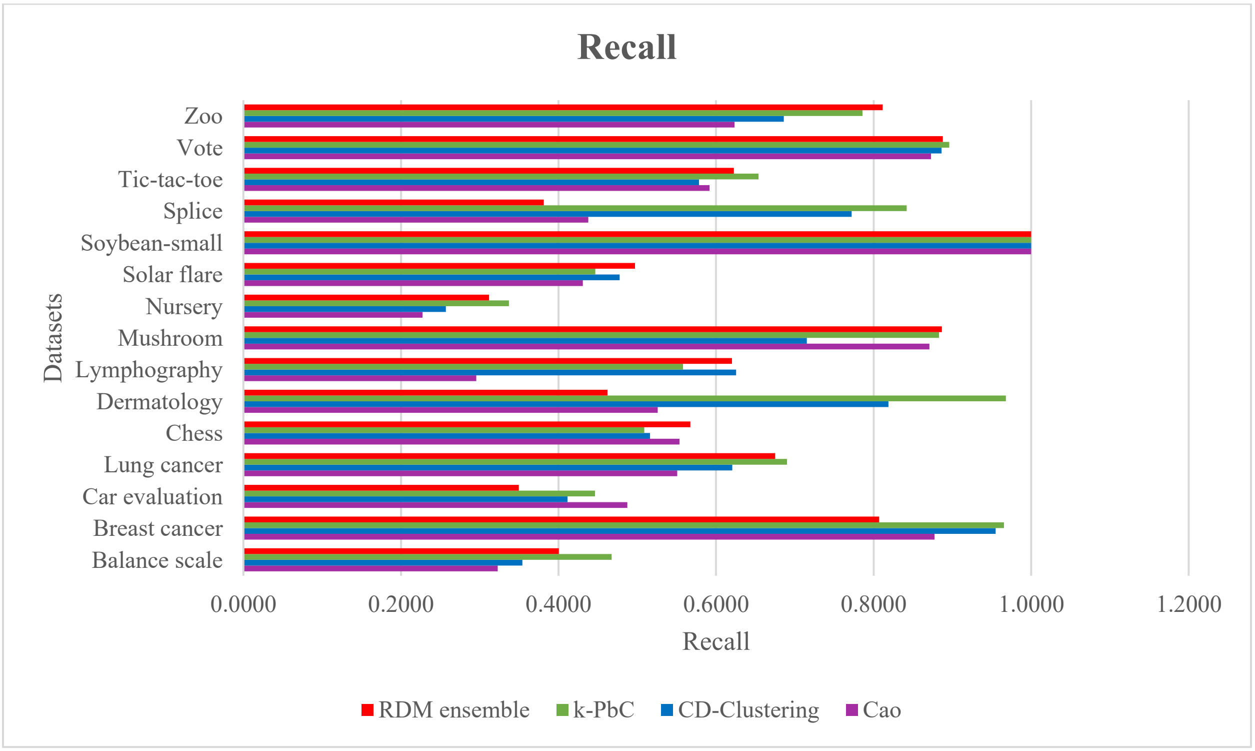

Figure 5.10 Recall of Compared Algorithms

As shown in Table 5.14 and Figure 5.10, k-PbC has the highest number of best recalls (9 datasets). RDM ensemble (with 5 datasets) stand in the second place, and CD-clustering and Cao (both with 2 datasets) stand in the third place.

As for average recall, k-PbC has the highest amount, CD-clustering and RDM ensemble are the next best ones, in order.

5.3.4. RDM Realization of EPP (RDM EPP)

In this section, we will see the elapsed time for each part of the RDM EPP (Figure 4.5) versus its sequential counterpart. The parallel elapsed times are the shortest times that have been measured while using a certain number of processes (*p*), and we refer to it as the best *p*.

We used 7 datasets for our measurements based on their sizes from small to large in order to present a better picture of how the size of a dataset will affect the speed-up.

To have a complete vision, we divided the elapsed time spent for each different clustering to complete, into two parts: initialization (RDM variants) and clustering (PV3). The initialization's elapsed times are the times that each different RDM variant takes to build the initial clusters, and the clustering's elapsed times are the times that the PV3 algorithm needs to cluster a dataset after starting off a certain set of initial clusters produced by each RDM variant. The ensemble time or the integration time (*t_ensemble*) is the same for both sequential and parallel. We did not parallelize the ensemble process because due to its simplicity, it is already very fast, and perhaps its parallelization benefits may not be worth it.

As already mentioned in Sections 4.5 and 4.6, RDM, SIG RDM, and SIG-SCAV RDM initializations have been parallelized, and we have measured the time they take for each dataset. On the other hand, GLBMD, SCAV, and PRM RDMs have not been parallelized, and we do not have any measured timings for them. This is not a problem because the most intensive and time-consuming calculations belong to SIG and SIG-SCAV RDMs. As a result, the maximum elapsed time is one of their timings. So, in an embarrassingly parallel framework as in EPP, it does not make any difference if we had the elapsed times for the three mentioned algorithms or not.

Datasets - #Instances	***Detailed Times (ms)***	**RDM**	**GLBMD**	**SCAV**	**PATT**	**SIG**	**SIG_SCAV**
Lymphography - 148	***t_initialization (seq)***	5	6	5	5	637	635
	t_initialization (par)	7	-	-	-	421	419
	t_clustering (seq)	18	27	23	18	89	31
	t_clustering (par)	26	18	10	25	18	18
	t_ensemble				1		
	T_seq	24	-	-	-	729	667
	T_par	34	-	-	-	440	438
	Speed-up	***0.71***	-	-	-	***1.65***	***1.52***

Table 5.15 Lymphography - RDM EPP for Best p = 5

Datasets - #Instances	***Detailed Times (ms)***	**RDM**	**GLBMD**	**SCAV**	**PATT**	**SIG**	**SIG_SCAV**
Vote - 434	***t_initialization (seq)***	11	12	11	11	195	199
	t_initialization (par)	8	-	-	-	76	79
	t_clustering (seq)	25	25	17	25	27	42
	t_clustering (par)	9	9	5	9	9	6
	t_ensemble	1					
	T_seq	37	-	-	-	223	242
	T_par	18	-	-	-	86	86
	Speed-up	***2.06***	***-***	***-***	***-***	***2.59***	***2.81***

Table 5.16 Vote - RDM EPP for Best p = 8

Datasets - #Instances	***Detailed Times (ms)***	**RDM**	**GLBMD**	**SCAV**	**PATT**	**SIG**	**SIG_SCAV**
Breast cancer - 698	***t_initialization (seq)***	20	21	20	20	11,075	10,978
	t_initialization (par)	13	-	-	-	5,834	5,796
	t_clustering (seq)	34	34	54	35	222	236
	t_clustering (par)	13	13	21	13	28	30
	t_ensemble	4					
	T_seq	58	-	-	-	11,301	11,218
	T_par	30	-	-	-	5,847	5,813
	Speed-up	***1.93***	***-***	***-***	***-***	***1.93***	***1.92***

Table 5.17 Breast cancer - RDM EPP for Best p = 8

Datasets - #Instances	***Detailed Times (ms)***	**RDM**	**GLBMD**	**SCAV**	**PATT**	**SIG**	**SIG_SCAV**
Car evaluation – 1,728	***t_initialization (seq)***	24	28	24	24	315	322
	t_initialization (par)	12	-	-	-	76	80
	t_clustering (seq)	135	136	157	146	259	201
	t_clustering (par)	20	24	75	31	73	65
	t_ensemble	10					
	T_seq	169	-	-	-	584	533
	T_par	42	-	-	-	159	155
	Speed-up	***4.02***	***-***	***-***	***-***	***3.67***	***3.44***

Table 5.18 Car evaluation - RDM EPP for Best p = 9

Datasets - #Instances	***Detailed Times (ms)***	**RDM**	**GLBMD**	**SCAV**	**PATT**	**SIG**	**SIG_SCAV**
Chess - 3196	***t_initialization (seq)***	183	204	174	187	11,388	11,397
	t_initialization (par)	100	-	-	-	2,997	2,945
	t_clustering (seq)	1,067	532	540	938	419	839
	t_clustering (par)	214	104	106	214	105	100
	t_ensemble				19		
	T_seq	1,269	-	-	-	11,826	12,255
	T_par	333	-	-	-	3,121	3,064
	Speed-up	***3.81***	**-**	**-**	**-**	***3.79***	***4.00***

Table 5.19 Chess - RDM EPP for Best p = 10

Datasets - #Instances	***Detailed Times (ms)***	**RDM**	**GLBMD**	**SCAV**	**PATT**	**SIG**	**SIG_SCAV**
Mushroom – 8,192	***t_initialization (seq)***	396	428	384	386	261,882	261,154
	t_initialization (par)	120	-	-	-	47,062	46,986
	t_clustering (seq)	825	819	825	1,449	4,000	2,788
	t_clustering (par)	123	121	164	218	410	290
	t_ensemble				52		
	T_seq	1,273	-	-	-	265,934	263,994
	T_par	175	-	-	-	47,225	47,302
	Speed-up	***7.27***	**-**	**-**	**-**	***5.60***	***5.58***

Table 5.20 Mushroom - RDM EPP for Best p = 10

Datasets - #Instances	***Detailed Times (ms)***	**RDM**	**GLBMD**	**SCAV**	**PATT**	**SIG**	**SIG_SCAV**
Nursery – 12,960	***t_initialization (seq)***	237	284	242	236	8,488	8,345
	t_initialization (par)	74	-	-	-	965	904
	t_clustering (seq)	2,248	2,271	2,266	2,227	2,235	2,264
	t_clustering (par)	382	385	342	275	381	387
	t_ensemble				85		
	T_seq	2,570	-	-	-	10,808	10,694
	T_par	467	-	-	-	1,431	1,376
	Speed-up	***5.50***	**-**	**-**	**-**	***7.55***	***7.77***

Table 5.21 Nursery - RDM EPP for Best p = 10

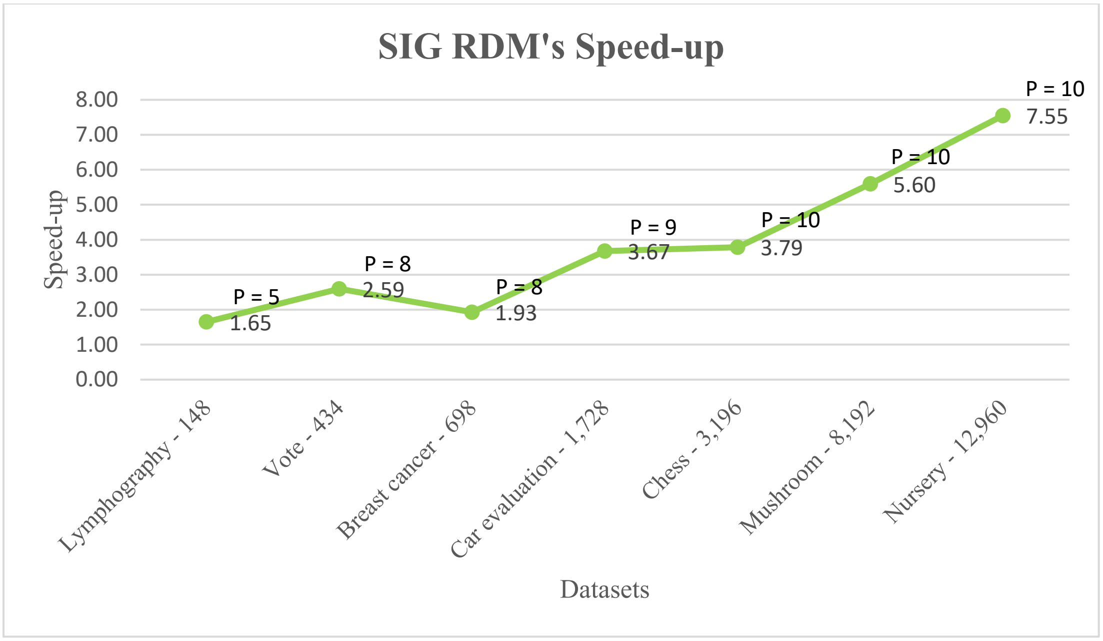

Figure 5.11 SIG RDM's Speed-up (a Close Approximation of RDM EPP's Speed-up)

Figure 5.11 shows the speed-up measurements in SIG RDM method while clustering 7 datasets with different sizes. We have chosen SIG RDM because it is one of the most time-consuming parts in RDM EPP. The other most time-consuming part is SIG-SCAV RDM which has an almost identical performance, in terms of speed-up, to SIG RDM. As said before, since EPP is an embarrassingly parallel framework, its total speed-up depends on its slowest component (SIG RDM). So, in reality, the total speed-up of the whole RDM EPP is almost equal to SIG RDM's speed-up (Figure 5.11).

As seen in Figure 5.11, the maximum speed-up of ***7.55*** obtained, was for the Nursery dataset with 12,960 instances (the largest dataset) using 10 processes while the least speed-up of ***1.65*** obtained, was for the smallest dataset, i.e., Lymphography with 148 instances using 5 processes. We can expect that when working with larger datasets, we obtain better speed-ups.

5.4. Summary

In this chapter, we first compared the performance of our different parallel algorithms. Quality-wise, PV3R stood first which was expectable. The reason is in the non-random initialization phase of this algorithm. With regard to speed-up, PV3 was the fastest among all. The superiority of PV3 over PV1 and PV3R was clear, and besides, it even outperformed PV2. This shows that speed-up can be affected by clustering quality too. Thus, if we use better algorithms (with less information loss and better, more effective communication between the processes), we can be hopeful that the overall amount of work may even decrease which in turn, helps with the clustering speed. On the other hand, with increasing the amount of communication between processes, our speed-up decreases. So, being able to maintain the quality while not losing much speed-up is similar to walking a tightrope and should be handled in such a way that these two do not interfere with one another and instead can help each other.

We then compared RDM and Cao's methods and showed that with our modification, better clustering quality was produced for most datasets. Then, by combining all our RDM variants' results in EPP framework (RDM EPP), RDM ensemble clustering results were produced which had comparable clustering quality to other state-of-the-art k-modes clustering algorithms, and stood among the top three best, to the best of our knowledge.

Finally, we presented the timing details of RDM EPP's different components. We saw that with increasing the size of the dataset, we can expect higher speed-ups in its most time-consuming parts, and as a result in RDM EPP, in general.

Chapter 6. Conclusion and Future Works

Better and faster tools are needed in this era of information explosion in order to gather knowledge from data and use their underlying potential. Clustering is a key intermediate step for many other applications to gain valuable insight. Most clustering algorithms do not work efficiently when dealing with large amounts of data. To add to the complexity, real-life data is mostly categorical or a mixture of categorical and numerical.

In this thesis, we tried to address the issues of clustering categorical data in an MPI parallel environment using k-modes algorithm. Dealing with categorical data is inherently challenging because of its nature as there is no straight-forward way of measuring the dissimilarity between two categorical data objects. Parallelizing it, on the other hand, exacerbate the whole situation because we cannot easily manage the k-modes' communications between the parallel components while not losing much speed-up. To tackle this problem, we proposed the PV3 algorithm after several attempts which could help us achieve this goal by benefiting from the idea of frequency tables.

By introducing RDM initialization algorithm, we could reach stability in our clustering quality, and by combining our RDM variants in an EPP framework with PV3 as the main vertical parallel clustering structure, we could maintain the quality while being faster. This implies that by replacing any state-of-the-art initialization techniques for k-modes instead of RDM, we can guarantee their sequential clustering quality while achieving speed-up too. On the other hand, the EPP framework is a generic consensus and embarrassingly parallel structure which provides us with flexibility in either the type of the clustering algorithms we use or the level of parallelism (Section 4.7).

In the future, and for the purpose of improving what have been done, this thesis problem can be looked at from two different angles: clustering quality and parallel speed-up.

As for clustering quality:

- we can point out the dissimilarity measure for categorical data. In our work, we have used others' methods with small or no changes for measuring distance between the categorical data objects, while new ways or even modifying the available ones remain to be explored.

- Another area that can be explored is the use of ensemble technique. As discussed in Section 4.7, we have integrated the end clustering results of different initialization/clustering algorithms to build the CR_{INT} in a voting manner (Figure 4.4). Another method of using the ensemble can be by integrating the initial clusters produced by different initialization algorithms *in the beginning*. In other words, we could do the voting process of ensemble *before* entering the clustering phase, build one set of integrated initial clusters, and then perform our clustering algorithm (k-modes in our case) on this set.

And as for parallel speed-up:

- We have used the frequency table concept in PV3 to decrease the amount of communication between MPI processes, other ideas or methods can be explored. For example, using gossip protocol comes across as an interesting topic in relation to k-modes clustering.

- Also, we can improve the k-modes algorithm's performance by calculating the modes and the frequency tables on the fly through updating them when adding a new object to each cluster as suggested by Tao in [29].

Due to the multidimensionality of the research problem, there are a great many ways to improve the parallel clustering of categorical data, some of which have been mentioned here. With the gained insight and knowledge from working on this subject, in the future, we would like to study GPU-Based parallelization of clustering categorical data.

Reference

[1] J. Chen, X. Lin, Q. Xuan, and Y. Xiang, "FGCH: a fast and grid based clustering algorithm for hybrid data stream," *Appl Intell*, vol. 49, no. 4, pp. 1228–1244, Apr. 2019, doi: 10.1007/s10489-018-1324-x.

[2] T. Deng, D. Ye, R. Ma, H. Fujita, and L. Xiong, "Low-rank local tangent space embedding for subspace clustering," *Information Sciences*, vol. 508, pp. 1–21, Jan. 2020, doi: 10.1016/j.ins.2019.08.060.

[3] M. Mojarad, S. Nejatian, H. Parvin, and M. Mohammadpoor, "A fuzzy clustering ensemble based on cluster clustering and iterative Fusion of base clusters," *Appl Intell*, vol. 49, no. 7, pp. 2567–2581, Jul. 2019, doi: 10.1007/s10489-018-01397-x.

[4] H. Wang, Y. Yang, B. Liu, and H. Fujita, "A study of graph-based system for multi-view clustering," *Knowledge-Based Systems*, vol. 163, pp. 1009–1019, Jan. 2019, doi: 10.1016/j.knosys.2018.10.022.

[5] Y. Zhang, Y. Yang, T. Li, and H. Fujita, "A multitask multiview clustering algorithm in heterogeneous situations based on LLE and LE," *Knowledge-Based Systems*, vol. 163, pp. 776–786, Jan. 2019, doi: 10.1016/j.knosys.2018.10.001.

[6] L. Bai, J. Liang, C. Dang, and F. Cao, "A cluster centers initialization method for clustering categorical data," *Expert Systems with Applications*, vol. 39, no. 9, pp. 8022–8029, Jul. 2012, doi: 10.1016/j.eswa.2012.01.131.

[7] D.-T. Dinh and V.-N. Huynh, "k-PbC: an improved cluster center initialization for categorical data clustering," *Appl Intell*, vol. 50, no. 8, pp. 2610–2632, Aug. 2020, doi: 10.1007/s10489-020-01677-5.

[8] S. S. Khan and A. Ahmad, "Cluster center initialization algorithm for K-modes clustering," *Expert Systems with Applications*, vol. 40, no. 18, pp. 7444–7456, Dec. 2013, doi: 10.1016/j.eswa.2013.07.002.

[9] F. Cao, J. Liang, and L. Bai, "A new initialization method for categorical data clustering," *Expert Systems With Applications*, vol. 36, no. 7, pp. 10223–10228, 2009, doi: 10.1016/j.eswa.2009.01.060.

[10] Z. Huang, "A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining," in *In Research Issues on Data Mining and Knowledge Discovery,* 1997, pp. 1–8.

[11] B. Barney, Livermore Computing (retired), D. Frederick and LLNL, "Introduction to Parallel Computing Tutorial," https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial##References (accessed Jan. 23, 2023).

[12] A. Grama, A. Gupta, G. Karypis and V. Kumar, "Analytical Modeling in Parallel Programs," in *introduction to Parallel Computing*, 2nd ed. Publisher: Addison Wesley, 2003, ch. 5, pp. 244.

[13] T. George and V. Sarin, "Domain Decomposition," in *Encyclopedia of Parallel Computing*, D. Padua, Ed. Boston, MA: Springer US, 2011, pp. 578–587. doi: 10.1007/978-0-387-09766-4_291.

[14] R. Buyya, C. Vecchiola, and S. T. Selvi, "Chapter 6 - Concurrent Computing: Thread Programming," in *Mastering Cloud Computing*, R. Buyya, C. Vecchiola, and S. T. Selvi,

Eds. Boston: Morgan Kaufmann, 2013, pp. 171–210. doi: 10.1016/B978-0-12-411454-8.00006-1.

[15] S. S. Khan and A. Ahmad, "Cluster Center Initialization for Categorical Data Using Multiple Attribute Clustering.," in *MultiClust@ SDM*, 2012, pp. 3–10.

[16] A. Ahmad and L. Dey, "A method to compute distance between two categorical values of same attribute in unsupervised learning for categorical data set," *Pattern Recognition Letters*, vol. 28, no. 1, pp. 110–118, Jan. 2007, doi: 10.1016/j.patrec.2006.06.006.

[17] P. Fournier-Viger, J. C.-W. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le, "A survey of itemset mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 4, p. e1207, 2017.

[18] Z. Dafir, Y. Lamari, and S. C. Slaoui, "A survey on parallel clustering algorithms for Big Data," *Artif Intell Rev*, vol. 54, no. 4, pp. 2411–2443, Apr. 2021, doi: 10.1007/s10462-020-09918-2.

[19] Z. He, S. Deng, and X. Xu, "Improving K-Modes Algorithm Considering Frequencies of Attribute Values in Mode," in *Computational Intelligence and Security*, vol. 3801, Y. Hao, J. Liu, Y. Wang, Y. Cheung, H. Yin, L. Jiao, J. Ma, and Y.-C. Jiao, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 157–162. doi: 10.1007/11596448_23.

[20] D. Dua and C. Graff, "UCI Machine Learning Repository." University of California, Irvine, School of Information and Computer Sciences, 2017. [Online]. Available: http://archive.ics.uci.edu/ml.

[21] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," Stanford, 2006.

[22] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable K-Means++." arXiv, Mar. 28, 2012. Accessed: Dec. 26, 2022. [Online]. Available: http://arxiv.org/abs/1203.6402

[23] O. M. San, V.-N. Huynh, and Y. Nakamori, "An alternative extension of the k-means algorithm for clustering categorical data," Int. J. Appl. Math. Comput. Sci., Vol. 14, No. 2, 241–247, 2004.

[24] T.-H. T. Nguyen, D.-T. Dinh, S. Sriboonchitta, and V.-N. Huynh, "A method for k-means-like clustering of categorical data," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–11, 2019.

[25] T.-H. T. Nguyen and V.-N. Huynh, "A k-Means-Like Algorithm for Clustering Categorical Data Using an Information Theoretic-Based Dissimilarity Measure," in *Foundations of Information and Knowledge Systems*, vol. 9616, M. Gyssens and G. Simari, Eds. Cham: Springer International Publishing, 2016, pp. 115–130. doi: 10.1007/978-3-319-30024-5_7.

[26] H. H. Nguyen, "Clustering Categorical Data Using Community Detection Techniques," *Computational Intelligence and Neuroscience*, vol. 2017, pp. 1–11, 2017, doi: 10.1155/2017/8986360.

[27] L. Chen and S. Wang, "Central Clustering of Categorical Data with Automated Feature Weighting,". in: IJCAI, 2013, pp 1260–1266.

[28] M. N. Joshi, "Parallel k-means algorithm on distributed memory multiprocessors," *Computer*, vol. 9, 2003.
[29] G. Tao, D. Xiangwu, and L. Yefeng, "Parallel k-modes algorithm based on MapReduce," in *2015 Third International Conference on Digital Information, Networking, and Wireless Communications (DINWC)*, Moscow, Russia, Feb. 2015, pp. 176–179. doi: 10.1109/DINWC.2015.7054238.

[30] Z. He, X. Xu, and S. Deng, "A cluster ensemble method for clustering categorical data," *Information Fusion*, vol. 6, no. 2, pp. 143–151, Jun. 2005, doi: 10.1016/j.inffus.2004.03.001.

[31] Z. He, X. Xu, and S. Deng, "Squeezer: An efficient algorithm for clustering categorical data," *J. Comput. Sci. & Technol.*, vol. 17, no. 5, pp. 611–624, Sep. 2002, doi: 10.1007/BF02948829.

[32] D. Cristofor and D. A. Simovici, "Finding median partitions using information-theoretical-based genetic algorithms.," *J. Univers. Comput. Sci.*, vol. 8, no. 2, pp. 153–172, 2002.
[33] A. Strehl and J. Ghosh, "Cluster ensembles—a knowledge reuse framework for combining multiple partitions," *Journal of machine learning research*, vol. 3, no. Dec, pp. 583–617, 2002.
[34] A. Strehl, *Relationship-based clustering and cluster ensembles for high-dimensional data mining*. The University of Texas at Austin, 2002.
[35] G. Karypis and V. Kumar, "A fast and high-quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.
[36] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proceedings of the 34th annual Design Automation Conference*, 1997, pp. 526–529.
[37] Z. Dafir, Y. Lamari, and S. C. Slaoui, "A survey on parallel clustering algorithms for Big Data," *Artif Intell Rev*, vol. 54, no. 4, pp. 2411–2443, Apr. 2021, doi: 10.1007/s10462-020-09918-2.

[38] D. S. Milojicic *et al.*, "Peer-to-peer computing." Technical Report HPL-2002-57, HP Labs, 2002.
[39] F. Jiang, G. Liu, J. Du, and Y. Sui, "Initialization of K-modes clustering using outlier detection techniques," *Information Sciences*, vol. 332, pp. 167–183, Mar. 2016, doi: 10.1016/j.ins.2015.11.005.

[40] L. Bai, J. Liang, C. Dang, and F. Cao, "A cluster centers initialization method for clustering categorical data," *Expert Systems with Applications*, vol. 39, no. 9, pp. 8022–8029, Jul. 2012, doi: 10.1016/j.eswa.2012.01.131.

[41] D. Xu and Y. Tian, "A Comprehensive Survey of Clustering Algorithms," Ann. Data. Sci., vol. 2, no. 2, pp. 165–193, Jun. 2015, doi: 10.1007/s40745-015-0040-1.

[1]

[42] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, “Density-based clustering in spatial databases: The algorithm gdbscan and its applications,” *Data mining and knowledge discovery*, vol. 2, pp. 169–194, 1998.

[43] Z. Wang, A. Xu, Z. Zhang, C. Wang, A. Liu, and X. Hu, “The Parallelization and Optimization of K-means Algorithm Based on Spark,” in 2020 15th International Conference on Computer Science & Education (ICCSE), Delft, Netherlands: IEEE, Aug. 2020, pp. 457–462. doi: 10.1109/ICCSE49874.2020.9201770.

[44] J. L. Gustafson, “Reevaluating Amdahl’s law,” Communications of the ACM, vol. 31, no. 5, pp. 532–533, 1988.

Appendix

Appendix A: Accuracy, Precision, and Recall of Compared Algorithms

Datasets	k-means++	k-means||	k-modes	k-reps	Cao	Khan	Mod-2	Mod-3	CD-Clustering	k-PbC	RDM ensemble	RDM
Balance scale	0.5168	0.5376	0.4497	0.4342	0.3760	0.4192	0.4310	0.4323	0.4129	**0.5680**	0.5224	0.4454
Breast cancer	0.9585	0.9585	0.7043	0.8949	0.9113	0.6323	0.9576	0.9570	0.9514	**0.9614**	0.8653	0.8653
Car evaluation	0.2789	0.3177	0.3592	0.3811	**0.4936**	0.3576	0.3725	0.3831	0.3125	0.3640	0.3652	0.3403
Lung cancer	0.5313	0.5938	0.5188	0.5400	0.5313	0.4375	0.5022	0.4922	0.5938	**0.6875**	**0.6875**	0.5938
Chess	0.5116	0.5116	0.5465	0.5367	0.5663	**0.7040**	0.5279	0.5385	0.5156	0.5047	0.5760	0.5673
Dermatology	0.7404	0.5874	0.5375	0.7161	0.5874	0.6175	0.7280	0.7404	0.8552	**0.9645**	0.6011	0.5109
Lymphography	0.3176	0.3919	0.4379	0.5301	0.3514	0.5068	0.5433	0.5341	0.5000	0.5946	**0.6081**	0.5135
Mushroom	0.7093	0.7093	0.7291	0.7897	0.8754	0.8288	0.7474	0.7682	0.7244	0.8861	**0.8902**	**0.8902**
Nursery	0.2409	0.2275	0.3324	0.3161	0.3673	0.2804	0.3165	0.3128	0.4156	**0.4492**	0.3613	0.3147
Solar flare	0.4540	0.4540	0.4524	0.5087	0.5432	**0.6463**	0.5526	0.5579	0.4343	0.5901	0.4925	0.4747
Soybean-small	0.7234	0.7447	0.7657	0.8834	**1.0000**	0.9787	0.9070	0.8666	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Splice	0.3937	0.3937	0.4054	0.5430	0.4009	0.4279	0.6496	0.6741	0.7260	**0.7746**	0.4169	0.3282
Tic-tac-toe	0.5585	0.5679	0.5675	0.5605	0.6106	0.6347	0.5625	0.5598	0.6044	0.6326	0.6263	**0.6837**
Vote	0.8690	0.5678	0.8622	0.8751	0.8644	0.8506	0.8764	0.8764	0.8713	**0.8805**	0.8733	0.8714
Zoo	0.7723	0.7129	0.6868	0.6986	0.6733	0.8614	0.7601	0.7524	0.8218	0.8911	**0.9063**	**0.9583**
Average Accuracy	0.572	0.552	0.557	0.614	0.610	0.612	0.629	0.630	**0.649**	**0.717**	**0.653**	0.624

Table 8.1 Accuracy of Compared Algorithms

Datasets	k-means++	k-means||	k-modes	k-reps	Cao	Khan	Mod-2	Mod-3	CD-Clustering	k-PbC	RDM ensemble	RDM
Balance scale	0.4896	**0.5394**	0.3993	0.4276	0.3282	0.3609	0.4177	0.4238	0.3751	0.4825	0.4624	0.4072
Breast cancer	**0.9571**	**0.9571**	0.7163	0.9182	0.9292	0.5535	0.9466	0.9459	0.9470	0.9517	0.9068	0.9068
Car evaluation	0.2789	0.2843	0.2868	0.3488	**0.3826**	0.2415	0.3407	0.3440	0.3125	0.3704	0.3381	0.3235
Lung cancer	0.5333	0.6902	0.5349	0.5937	0.5468	0.4468	0.5726	0.5515	0.5980	**0.7421**	0.7269	0.6435
Chess	0.5110	0.5110	**0.7018**	0.5416	0.5796	0.5312	0.5326	0.5425	0.5162	0.5095	0.5799	0.5699
Dermatology	0.7583	0.5782	0.5077	0.6603	0.5604	0.6841	0.6696	0.6589	0.7543	**0.9612**	0.3880	0.3877
Lymphography	0.3147	0.3378	0.3906	0.4659	0.2698	0.4226	0.4750	0.4599	0.4724	**0.4752**	0.4423	0.4281
Mushroom	0.7740	0.7740	0.7496	0.8018	0.9019	0.8688	0.7586	0.7781	0.7990	0.9000	**0.9056**	**0.9056**
Nursery	0.2323	0.2195	0.2905	0.2995	0.2978	0.2304	0.2954	0.2930	**0.6494**	0.4352	0.3589	0.3062
Solar flare	0.3381	0.3381	0.3355	0.3400	0.3381	0.3361	0.3415	0.3403	0.3377	**0.3425**	0.3397	0.3384
Soybean-small	0.7569	0.7708	0.7576	0.8769	**1.0000**	0.9773	0.9060	0.8522	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Splice	0.3377	0.3377	0.4095	0.5960	0.4518	0.4260	0.6655	0.6898	0.7198	**0.7815**	0.3855	0.3011
Tic-tac-toe	0.5509	0.5509	0.5540	0.5521	0.5859	0.6071	0.5604	0.5597	0.5746	0.6396	0.6123	**0.6635**
Vote	0.8636	0.5433	0.8573	0.8705	0.8568	0.8484	0.8724	0.8724	0.8670	**0.8762**	0.8684	0.8668
Zoo	0.6948	0.5383	0.5523	0.5788	0.5996	0.7390	0.6518	0.6193	0.5688	0.7503	0.7465	**0.9472**
Average Precision	0.559	0.531	0.536	0.591	0.575	0.552	0.600	0.595	**0.633**	**0.681**	**0.604**	0.600

Table A.2 Precision of Compared Algorithms

Datasets	k-means++	k-means||	k-modes	k-reps	Cao	Khan	Mod-2	Mod-3	CD-Clustering	k-PbC	RDM ensemble	RDM
Balance scale	0.4585	**0.4905**	0.3852	0.4039	0.3228	0.3541	0.3957	0.4009	0.3540	0.4673	0.4005	0.3952
Breast cancer	0.9506	0.9506	0.6951	0.8535	0.8773	0.5336	0.9637	0.9632	0.9550	**0.9656**	0.8071	0.8539
Car evaluation	0.3554	0.2960	0.3034	0.3794	**0.4875**	0.2499	0.3650	0.3639	0.4114	0.4465	0.3498	0.3436
Lung cancer	0.5581	0.6168	0.5350	0.5380	0.5507	0.4470	0.4595	0.4512	0.6208	**0.6900**	0.6749	0.5342
Chess	0.5110	0.5110	**0.6962**	0.5389	0.5537	0.5540	0.5296	0.5394	0.5162	0.5091	0.5673	0.4858
Dermatology	0.7649	0.5182	0.4735	0.6811	0.5260	0.6165	0.6960	0.6943	0.8189	**0.9679**	0.4620	0.3758
Lymphography	0.3425	0.3765	0.4569	0.5746	0.2955	0.4451	0.5438	0.5328	**0.6255**	0.5579	0.6204	0.4154
Mushroom	0.7002	0.7002	0.7256	0.7858	0.8709	0.8228	0.7472	0.7686	0.7151	0.8829	**0.8868**	0.8833
Nursery	0.2061	0.1885	0.2581	0.2551	0.2273	0.2044	0.2516	0.2426	0.2569	**0.3370**	0.3117	0.2097
Solar flare	0.4841	0.4841	0.3924	0.4884	0.4310	0.4379	0.4923	0.4807	0.4775	0.4467	**0.4970**	**0.5002**
Soybean-small	0.7574	0.7721	0.7691	0.8786	**1.0000**	0.9853	0.9006	0.8567	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Splice	0.3366	0.3366	0.4205	0.6163	0.4379	0.4472	0.7015	0.7291	0.7722	**0.8421**	0.3815	0.4044
Tic-tac-toe	0.5560	0.5555	0.5579	0.5559	0.5917	0.6129	0.5652	0.5644	0.5785	**0.6538**	0.6228	0.5552
Vote	0.8822	0.5431	0.8751	0.8898	0.8730	0.8672	0.8921	0.8920	0.8863	**0.8960**	0.8880	0.8914
Zoo	0.7080	0.5498	0.5936	0.6129	0.6233	0.7648	0.6503	0.6494	0.6860	0.7860	**0.8115**	0.7662
Average Recall	0.571	0.526	0.543	0.603	0.578	0.556	0.610	0.609	**0.645**	**0.697**	**0.619**	0.574

Table 8A8.3 Recall of Compared Algorithms