ROYAUME DU MAROC
Université Mohammed V - RABAT -
جامعة محمد الخامس – الرباط –
Faculté des sciences
كلية العلوم
CENTRE D'ETUDES DOCTORALES - SCIENCES ET TECHNOLOGIES

N° d'ordre : 3711

THESE

En vue de l'obtention du : ***DOCTORAT***

Structure de Recherche *: ANISSE*
Discipline *: Informatique*
Spécialité *: Intelligence Artificielle et Big Data*

Présentée et soutenue le : 26/11/2022 par :

Zineb DAFIR

Design and Development of new Heuristics for Clustering Big Data based on Artificial Intelligence

JURY

Mohammed BENKHALIFA	PES, Université Mohammed V, Faculté des sciences de Rabat	Président/Rapporteur
Rochdi MESSOUSSI	PES, Université Ibn Tofail, Faculté des sciences de Kénitra	Rapporteur/Examinateur
Houda BENBRAHIM	PES, Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes, Rabat	Rapporteuse/Examinatrice
Fouzia OMARY	PES, Université Mohammed V, Faculté des sciences de Rabat	Examinatrice
Said SLAOUI	PES, Université Mohammed V, Faculté des sciences de Rabat	Directeur de thèse

Année Universitaire: 2021/2022

Dedication and acknowledgements

It is a pleasure for me to express my profound thanks to my dear mother and father for their unlimited moral support and their presence and encouragement throughout my studies. I also thank my husband, Mohammed, for his continued support, help, and encouragement throughout the preparation process. I want to thank my family members equally for their endless encouragement, namely my brothers Amine and Abderrahmane, and my lovely sisters Asmae and Wiam. I would like to thank my best friends and colleagues at Mohammed V for their encouragement and love. Eventually, this work will also be dedicated to my hero, Omar, and my little princess, Nada.

ACKNOWLEDGEMENTS

This thesis was conducted in the computer science department within the research team: Algorithms, Networks, Intelligent Systems and Software Engineering(ANISSE) at the Faculty of Sciences of Rabat.

First and foremost, I'd like to thank my thesis advisor, Professor Mr. **Said SLAOUI** (PES) at the Faculty of Sciences, Mohammed V University in Rabat, for directing and supervising this work with great scientific rigor. The quality of his advice, the support, and the confidence he gave me allowed me to develop my ideas and give me another vision of research. These years spent with him on this team have given me an incredibly enriching experience, both in study and teaching.

I would like to thank Professor Mr. **Mohammed BENKHALIFA** (PES) at the Faculty of Sciences in Rabat and the director of the research team ANISSE for his encouragement during my doctoral research and studies. I also want to thank him for agreeing to take part in my thesis committee as President of the Jury and for dedicating his precious time to read my dissertation and enrich my research work with his valuable comments.

A special thanks to Professor Mr. **Rochdi MESSOUSSI** (PES) at the Faculty of Sciences of Kénitra, University Ibn Tofail of Kénitra, for giving me the honor of being a reporter for this research work and for participating in the defense of my thesis with his precious comments.

Furthermore, I would like to thank Professor Mrs. **Houda BENBRAHIM** (PES) from the Department of Computer Science and Decision Support at the National School of Computer

Science and Systems Analysis (ENSIAS), University Mohammed V of Rabat, for having accepted to review my dissertation and for having devoted her precious time to examining this manuscript.

Moreover, I want to thank Professor Mrs. **Fouzia OMARY** (PES) at the Faculty of Sciences of Rabat and the director of the IPSS laboratory for her encouragement during my doctoral research and studies, and for agreeing to participate in the defense of my thesis with her very valuable comments.

Abstract

The exponential growth in the use of digital technologies in almost all areas of the real world, has given rise to very large quantities of data of various types, usually with complex structures. The emergence of Big Data technology is pushing researchers to invent new approaches, sophisticated artificial intelligence techniques, and powerful algorithms exploiting Big Data platforms to keep abreast of the advancements related to this new concept. In particular, clustering is one of the relevant data mining tasks, which aims to process Big Data sets effectively. This thesis focuses on the development of new heuristics for clustering Big Data based on Artificial Intelligence in order to address some of the issues raised in the literature. In this regard, the first suggested contribution concerns a technical overview of the latest parallel clustering algorithms categorized according to the computing platforms. The second contribution consists of developing a clustering heuristic called E-Transitive, an enhanced version of the Transitive heuristic, designed to process categorical data sets based on the Relational Analysis approach. The third proposal is a parameter-free clustering algorithm based on k-means (PFK-means), aiming to generate the initial cluster centers progressively until the appropriate number of clusters is automatically detected. The last contribution aims to provide a new parallel clustering algorithm based on Spark, designed to enhance the sequential PFK-means algorithm for the purpose of leveraging distributed systems and processing Big Data. The experiments conducted on both, real-life and synthetic data sets, demonstrated that the proposed methods yield good quality results in reasonable computational time without resorting to complex input parameters.

Keywords: Artificial Intelligence, Clustering, Categorical data, Quantitative data, Big data, k-means, MapReduce, Spark framework, Parallel computing.

RÉSUMÉ

La croissance exponentielle de l'utilisation des technologies numériques dans presque tous les domaines du monde réel, a donné naissance à de très grandes quantités de données de différents types, généralement avec des structures complexes. L'émergence de la technologie Big Data pousse les chercheurs à inventer de nouvelles approches, des techniques d'intelligence artificielle sophistiquées et des algorithmes puissants exploitant les plateformes Big Data pour suivre les avancées liées à ce nouveau concept.

En particulier, le clustering est l'une des tâches pertinentes de l'exploration de données, qui vise à traiter efficacement les ensembles de Big Data. Cette thèse se concentre sur le développement de nouvelles heuristiques pour le clustering de Big Data basées sur l'Intelligence Artificielle afin de répondre à certaines des questions soulevées dans la littérature. A cet égard, la première contribution proposée concerne un aperçu technique des derniers algorithmes de clustering parallèles catégorisés en fonction des plateformes de calcul. La deuxième contribution consiste à développer une heuristique de clustering appelée E-Transitive, une version améliorée de l'heuristique Transitive, conçue pour traiter des ensembles de données catégoriques basés sur l'approche de l'analyse relationnelle. La troisième proposition est un algorithme de clustering sans paramètre basé sur les k-means (PFK-means), visant à générer les centres de clusters initiaux progressivement jusqu'à ce que le nombre approprié de clusters soit automatiquement détecté. La dernière contribution vise à fournir un nouvel algorithme de clustering parallèle basé sur Spark, conçu pour améliorer l'algorithme séquentiel PFK-means dans le but d'exploiter les systèmes distribués et de traiter les Big Data. Les expériences menées sur des ensembles de données réelles et synthétiques ont démontré que les méthodes proposées donnent des résultats de bonne qualité dans un temps de calcul raisonnable sans recourir à des paramètres d'entrée complexes.

Mots-clés : Intelligence artificielle, clustering, données catégoriques, données quantitatives, Big data, k-means, MapReduce, Spark framework, Parallel computing.

Résumé Détaillé

Ces dernières années, le monde a connu une évolution rapide dans tous les domaines, en particulier dans les technologies numériques, qui ont provoqué la génération d'une grande quantité de données provenant de nombreuses sources, notamment les réseaux sociaux, l'internet des objets, les applications mobiles, le multimédia, les services financiers, les systèmes ERP et de nombreuses autres sources dépendant de technologies numériques sophistiquées. Ce progrès sans précédent a fait naître de nouveaux concepts tels que le Big Data et l'intelligence artificielle qui sont extrêmement liés, de sorte que nous ne pouvons pas extraire les bonnes connaissances de ces données pour prendre des décisions sans utiliser l'intelligence artificielle. D'autre part, le Big Data est une partie fondamentale du processus d'apprentissage des modèles d'intelligence artificielle et donc la fiabilité et la disponibilité des données collectées sont cruciales pour le modèle intelligent choisi. Ainsi, la génération continue de données énormes, variées et complexes nécessite l'amélioration et l'adaptation des outils d'intelligence artificielle pour suivre cette avancée et répondre aux besoins des entreprises et des industries.

En particulier, l'analyse de clusters est une tâche d'exploration de données permettant le traitement efficace des Big Data. L'objectif de l'analyse en cluster est de placer les objets de données similaires dans le même groupe et les objets dissemblables dans des groupes différents pour construire des clusters disjoints. La similarité entre les objets de données est définie par une mesure de distance qui dépend fortement du type d'ensemble de données à traiter et des caractéristiques définissant l'ensemble de données. Par conséquent, différents algorithmes de clustering peuvent donner des résultats très différents pour le même ensemble de données. De plus, le clustering des Big data est basé sur le concept de calcul parallèle réalisé par des plateformes de mise à l'échelle horizontale ou verticale. En d'autres termes, ces plates-formes aux architectures différentes visent soit à distribuer les tâches de clustering sur plusieurs nœuds qui représentent

des machines de calcul, soit à utiliser des serveurs dotés d'une grande capacité de calcul et de stockage pour effectuer les calculs appropriés.

À cet égard, plusieurs algorithmes de clustering ont été conçus au fil des ans afin de traiter efficacement ces grandes quantités de données provenant de sources diverses. Cependant, malgré la contribution significative apportée par ces algorithmes, ils présentent toujours de sérieux défis concernant plusieurs aspects parmi lesquels, la connaissance préalable de certains paramètres d'initialisation, l'ordonnancement des données, les paramètres d'entrée complexes, et l'extensibilité de ces algorithmes pour le traitement de données larges et variées.

Afin de surmonter ces problèmes, cette thèse se concentre sur la conception et le développement de nouvelles heuristiques pour le clustering de Big data. Ainsi, la première contribution présente une vue d'ensemble des derniers algorithmes de clustering parallèle classés en fonction des plates-formes informatiques utilisées pour traiter les Big Data, à savoir les plates-formes à échelle horizontale et verticale. La première catégorie comprend les réseaux peer-to-peer, les plateformes MapReduce et Spark, tandis que la seconde comprend les processeurs multicoeurs, les unités de traitement graphique et les plateformes Field Programmable Gate Arrays. En outre, il comprend une comparaison des performances des algorithmes examinés sur la base de certains critères communs de validation du regroupement dans le contexte du Big Data. La deuxième contribution consiste à améliorer l'heuristique $Transitive$ qui vise à traiter de grands ensembles de données catégorielles, en exploitant les avantages offerts par l'approche de l'Analyse Relationnelle. En effet, cette heuristique détecte automatiquement le nombre maximal de clusters et le seuil de similarité. Les qualités distinctives de l'heuristique $Transitive$ incluent le fait qu'il s'agit d'une méthode à paramètres libres qui effectue le clustering sans spécifier le nombre de clusters. En outre, elle adopte une structure de cluster spécifique afin de réduire le temps de calcul. Un autre avantage de cette méthode est sa capacité à diminuer le coût des calculs et donc, à limiter la consommation de mémoire. L'heuristique proposée concerne certaines faiblesses de l'heuristique $Transitive$ originale, à savoir le calcul du représentant et la manière dont chaque objet de données est traité. Cette thèse aborde également le problème de la fixation des paramètres d'entrée de l'algorithme k-means qui affecte la performance et la qualité du clustering. Par conséquent, elle vise à utiliser les forces de l'heuristique E-Transitive pour découvrir automatiquement le

nombre de clusters et les centres initiaux des clusters en développant une nouvelle méthode de clustering nommée PFK-means basée sur l'algorithme k-means. L'heuristique proposée est un algorithme de clustering sans paramètre puisqu'il ne nécessite pas l'initialisation préalable du nombre de clusters. Ainsi, elle génère progressivement les centres de clusters initiaux jusqu'à ce que le nombre approprié de clusters soit automatiquement détecté. Ainsi, les améliorations obtenues grâce à cette heuristique concernent principalement deux faiblesses importantes de l'algorithme k-means. La première consiste à fixer le nombre de clusters k, et la seconde se concentre sur la détermination des centres de clusters initiaux. Ce travail se concentre également sur les outils de calcul parallèle tels que le cadre Spark basé sur le modèle MapReduce et la façon dont il peut être exploité pour traiter les Big Data. Il vise donc à améliorer les performances de l'algorithme PFK-means en développant un nouvel algorithme de regroupement parallèle utilisant le cadre Spark et un modèle basé sur l'apprentissage automatique. Plus précisément, l'algorithme proposé vise à paralléliser les deux principales étapes de l'algorithme PFK-means en utilisant des fonctions RDDs de Spark. La première consiste à appliquer des fonctions RDDs successives afin d'effectuer le calcul des distances euclidiennes, de construire les clusters initiaux, et enfin de mettre à jour les différents clusters pour obtenir les centres de clusters et, par conséquent, le nombre k. La deuxième étape fait appel à l'utilisation du K-means parallèle fourni par Spark mLlib avec comme paramètres d'entrée les centres de clusters et le nombre de clusters découverts dans la phase précédente. Mots-clés : Intelligence artificielle, clustering, données catégoriques, données quantitatives, Big data, k-means, MapReduce, Spark framework, Parallel computing

TABLE OF CONTENTS

List of Tables

List of Figures

Figure **Page**

CHAPTER 1

GENERAL INTRODUCTION

1.1 Context and Problematic issues

In recent years, the world has witnessed a rapid evolution in all fields, especially in digital technologies, which have caused the generation of a large amount of data coming from many sources, including social networks, the internet of things, mobile apps, multimedia, financial services, ERP systems, and many other sources depending on sophisticated digital technologies. This unprecedented progress has raised new concepts such as Big Data and Artificial Intelligence that are extremely interrelated so that we cannot extract the right knowledge from these data to make decisions without using artificial intelligence. On the other hand, Big data is a fundamental part of the learning process of artificial intelligence models and therefore the reliability and availability of the collected data are crucial for the chosen intelligent model. Thus, the continuous generation of huge, varied, and complex data requires the improvement and adaptation of artificial intelligence tools to keep up with this advancement and to meet the needs of businesses and industries.

In particular, cluster analysis is a data-mining task allowing the efficient processing of Big Data. The aim of cluster analysis is to place similar data objects in the same group and the dissimilar ones into different groups to build disjoint clusters. The similarity between data objects is defined by a distance measure which depends strongly on the type of the data set to be

processed and the features defining the data set. Consequently, different clustering algorithms can yield significantly different results for the same data set. Furthermore, clustering Big data is based on the concept of parallel computing realized by horizontal scaling platforms or vertical scaling platforms. In other words, these platforms with different architectures aim either at distributing the clustering tasks over several nodes that represent computing machines, or at using servers with a high computing and storage capacity to perform the appropriate calculations. In that respect, several clustering algorithms have been designed over the years in order to efficiently deal with these large amounts of data from various sources. However, despite the significant contribution provided by these algorithms, they still present serious challenges concerning several aspects among which, prior knowledge of some initialization parameters, data ordering, complex input parameters, and the scalability of these algorithms for processing large and varied data.

In order to overcome these issues, this thesis focuses on the design and development of new heuristics for Big data clustering. Thus, firstly, this research consists in improving the $Transitive$ heuristic [117] allowing it to deal with large categorical data based on the Relational Analysis approach. This research also addresses the problem of fixing input parameters of the k-means algorithm which affects the performance and quality of clustering. Therefore, it aims to use the strengths of the E-Transitive heuristic to discover the number of clusters and the initial cluster centers automatically by developing a new clustering method named PFK-means based on the k-means algorithm. This work also focuses on parallel computing tools such as the Spark framework based on the MapReduce model and how it can be exploited to process Big Data. Hence, it intends to improve the performance of the PFK-means algorithm by developing a new parallel clustering algorithm using the Spark framework and a machine learning-based model.

1.2 Background and Basic Concepts

The work developed in this thesis is mainly focused on clustering Big Data using artificial intelligence. This part explores the different approaches related to Big Data, including the definition of the Big Data concept, the different sources of Big Data, the Big Data life cycle, and

the categories of Big Data platforms. This part also includes the definition of cluster analysis, its requirements, the different clustering methods, and the cluster analysis applications.

1.2.1 Big Data

1.2.1.1 Definition

The development of the digital world allows the generation of huge amounts of data including messages, videos and photos collected from various social networks. A study on Big Data has proven that 2.5 quintillion (2,500,000,000,000,000, or 2.5 × 1018) bytes of data are collected every day. This means a huge amount of data. Big data therefore covers complex data that is too large, not necessarily structured, and arrives at high speed. Thus, these data are delicate to handle by traditional database management systems. This requires Big data technologies to be able to collect, store, process, and interpret the data in distributed machines. Hadoop MapReduce is an example of Big data technology using HDFS for data storage and MapReduce as a processing model[15]. Figure 1.1 shows the evolution of Big Data over the years. Big Data is characterized

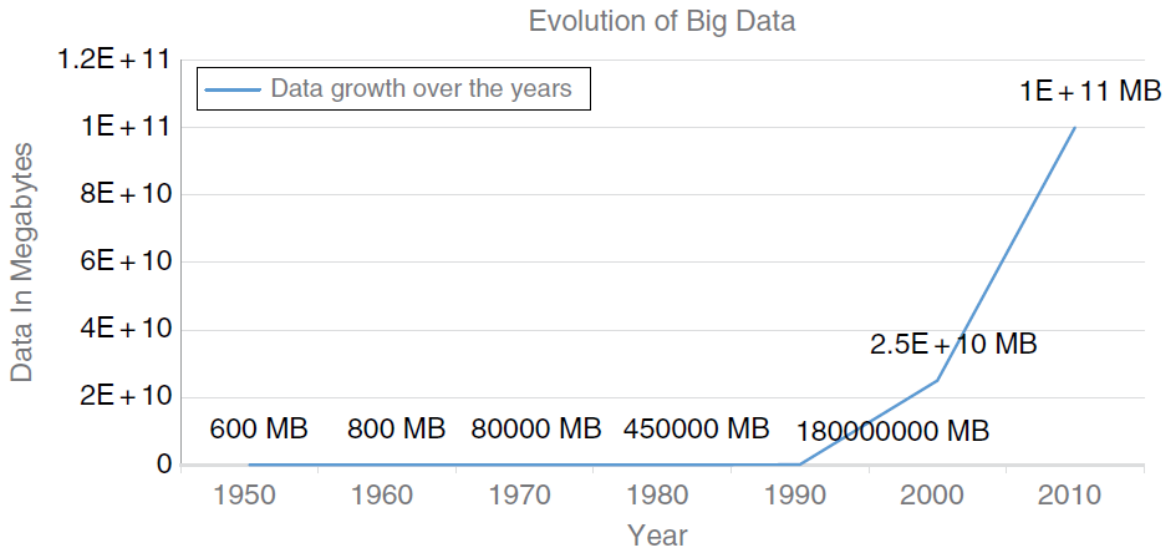

Figure 1.1: Evolution of Big Data [15].

by three different dimensions: The size of the data known as " volume ", the heterogeneity of the data called " variety ", and the rate at which the data is generated and processed, known as " velocity ".

Volume

The exceptional growth in data volume is due to the significant use of social networks, point-of-sale (POS) transactions, online banking and GPS sensors. Pushing companies and industries to better develop business solutions to meet customers' needs by improving the quality of services. Big data volume measures from terabytes to zettabytes (1024 GB = 1 terabyte; 1024 TB = 1 petabyte; 1024 PB = 1 exabyte; 1024 EB = 1 zettabyte; 1024 ZB = 1 yottabyte)[15].

Velocity

Velocity refers to the speed with which data is generated, processed, and analyzed. But, the exponential evolution that Big Data has known has caused the generation of data with a very high speed that is sometimes difficult to capture, and consequently, they are not analyzed [15]. Figure 1.2 shows the high-velocity of data sets provided online in 60 seconds.

Variety

The data produced by the various sources of Big Data has three main formats: structured, semi-structured, and unstructured. Structured data is data handled by standard database systems. Semi-structured data is a mixture of structured and unstructured data such as XML data. Unstructured data is data that does not have a precise structure including electronic messages, photos, and web pages.

1.2.1.2 Sources of Big Data

The emergence of the Big Data approach coincided with the technological revolution that caused the digitization of almost all real-world services, which consequently multiplied the sources of Big Data. These new services include electronic bill payment, online shopping, communication through social media, email transactions in various organizations, digital representation of organizational data, etc [15]. This section highlights some of the more popular big data sources.
Sensors: these include accelerometric sensors that are embedded in mobile devices, proximity sensors, and sensors in vehicles and medical devices.
Healthcare: The sources of Big Data in healthcare encompass three categories. Electronic medical

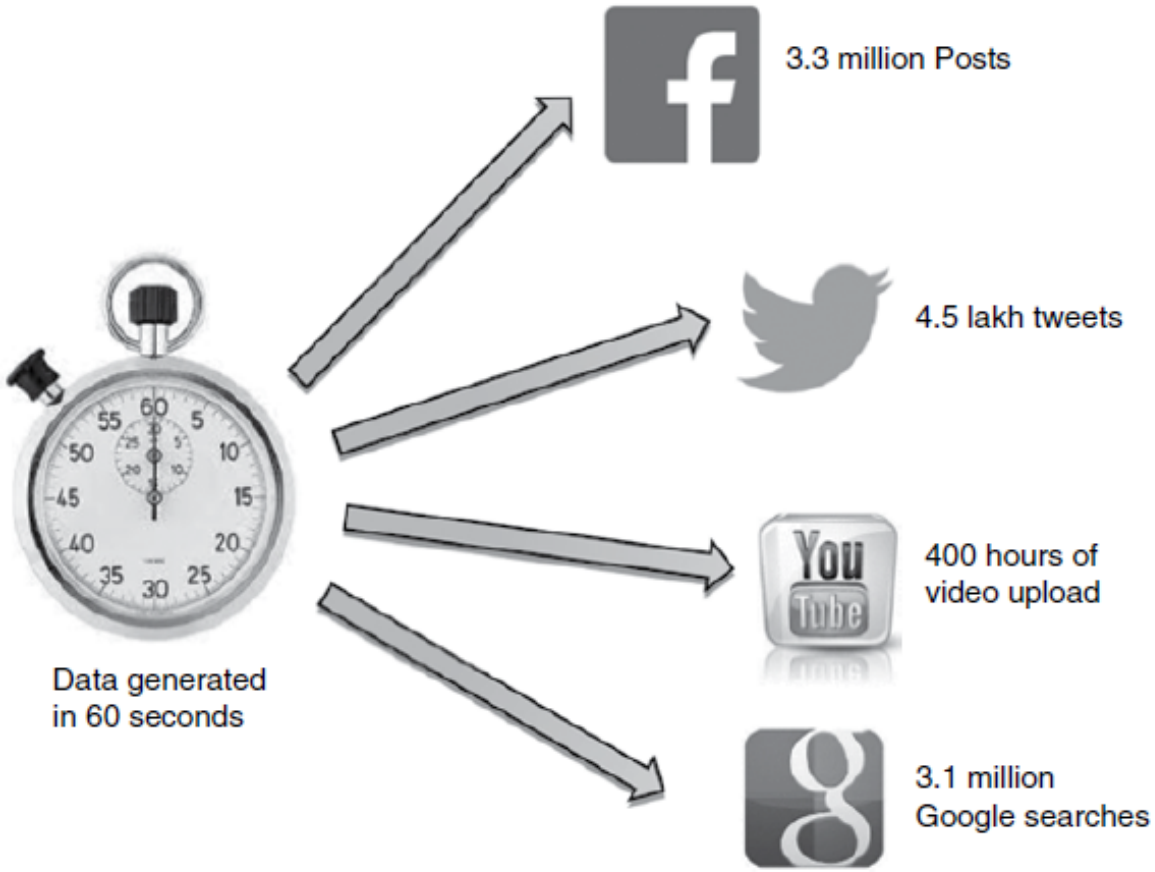

Figure 1.2: High-velocity data sets generated online in 60 seconds. [15].

records that include all patient information, portals for patients to access their personal medical records, and the clinical data repository.

Black Box: The black box captures flight operations, crew calls, and performance information for aircraft, helicopters, and jets.

Web Data: Web data is generated by the flow of clicks on the links of the websites that constitute the information of the customers' purchase transactions, their interests and habits. The analysis of these data helps to produce appropriate recommendations to consumers and display pertinent advertisements.

Organizational data: Organizational data consists of email transactions and documents produced in organizations.

Figure 1.3 lists the different data sources.

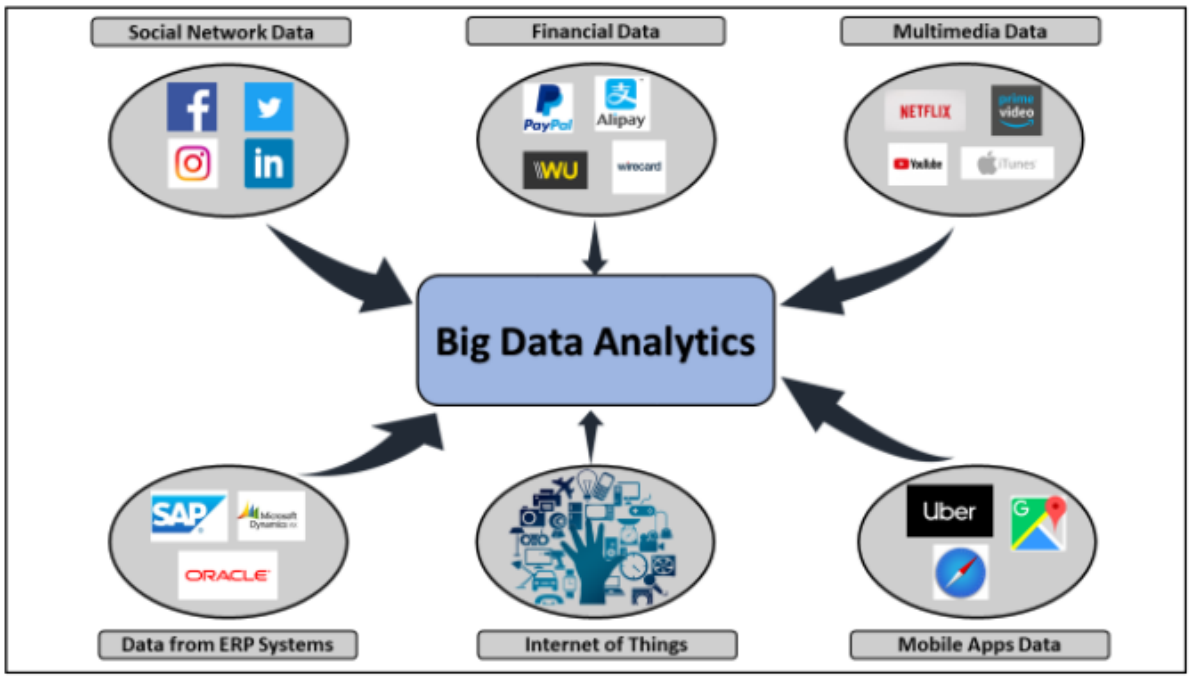

Figure 1.3: Examples of Big Data sources [78]

1.2.1.3 Big Data Life Cycle

Since the data sources are multiple, as explained in the previous section, the formats of the captured data and their speed of arrival vary depending on the source. The Big Data infrastructure includes new computing models capable of handling distributed and parallel processing operations with highly scalable storage and performance. The data is then collected and stored in a storage platform such as HDFS or NoSQL and then passed on to the analytical layer, which extracts the hidden knowledge using Big Data tools such as MapReduce and Yarn. The transition from storage to analysis requires a preprocessing step. The data therefore undergoes a complete cycle, starting from data generation, acquisition, preprocessing, integration, cleaning, transformation, analysis, and ending with visualization, aiming to facilitate decision-making [15]. Figure 1.4 illustrates the life cycle of Big Data. The various phases of the Big Data life cycle are presented below.

Big Data Generation

The first phase of the Big Data life cycle is the creation of various data from multiple sources at a

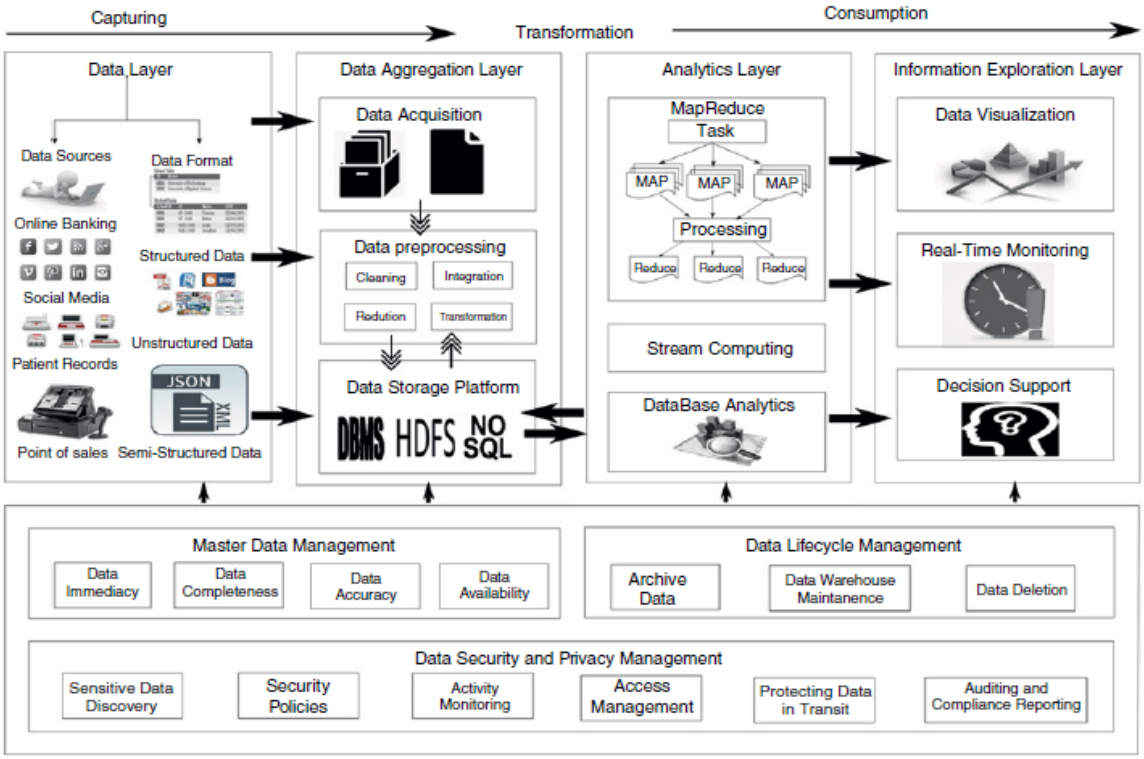

Figure 1.4: Big Data Life Cycle [15]

high speed that is continuously increasing. This data is in very large quantities and presented in different formats.

Data Aggregation

This phase follows the generation phase, which consists of collecting raw data at a huge volume, and at very high speed. Then this data is transferred to storage platforms that handle the processing and various analytical applications. The collected data can present errors, duplications that increase the cost of storage, hence the need for a long pre-processing process to obtain consistent, reliable and accurate data without errors.

Data Preprocessing

The preprocessing phase is essential to obtaining consistent and accurate data. It consists of four main steps. The first step is data integration, which consists of combining structured, unstructured, and semi-structured data from multiple sources to prepare it for cleaning. Subsequently,

the cleaning step focuses on a detailed analysis of the heterogeneous data to detect and determine errors that will be subsequently rectified, or completely removed. Missing values are filled by a global constant. While noisy data is processed by suitable methods including regression, clustering, binning and manual inspection. The third step in the preprocessing process is the reduction of the data volume by suitable techniques that maintain the integrity of the current data. These reduction techniques include data compression, dimensionality reduction, and numerical reduction. The final step is to transform the data into coherent and understandable information using ETL tools. This step uses four strategies involving smoothing, aggregation, generalization, and discretization.

Big Data Analytics

This phase consists of analyzing the pre-processed data with sophisticated tools and technologies that adapt to the continuous growth of the volume of data generated. The goal of this analysis is to create business value by making the right decisions according to the application domain. There are three types of analytics, including descriptive analytics, predictive analytics and perspective analytics.

Visualizing Big Data

Visualization is the last phase of the Big Data life cycle, providing a comprehensible representation of the heterogeneous data collected in the first phase. Thus, it simplifies knowledge extraction and, consequently, decision making. Among the visualization techniques, line graphs, bar charts, scatter diagrams, bubble charts and pie charts.

1.2.1.4 Big Data platforms

With the purpose of defining different platforms for Big Data analytics, we chose a new taxonomy proposed in [112]. According to this classification, Big Data platforms can be divided into two categories: horizontally scaling platforms and vertically scaling platforms. The first category includes systems that spread the workload over many servers or commodity machines. Therefore, it includes peer-to-peer networks, MapReduce, and Spark platforms. The second category

summarizes systems that run on a single server and can add additional resources such as processors, memory, and faster hardware. This category includes high-performance computing clusters (HPC), multicore processors, graphics processing units (GPUs), and field programmable gate array (FPGA) platforms. Figure 1.5 shows the taxonomy of different Big Data platforms. The description of the Big Data platform in this section is part of the original contribution published on [37].

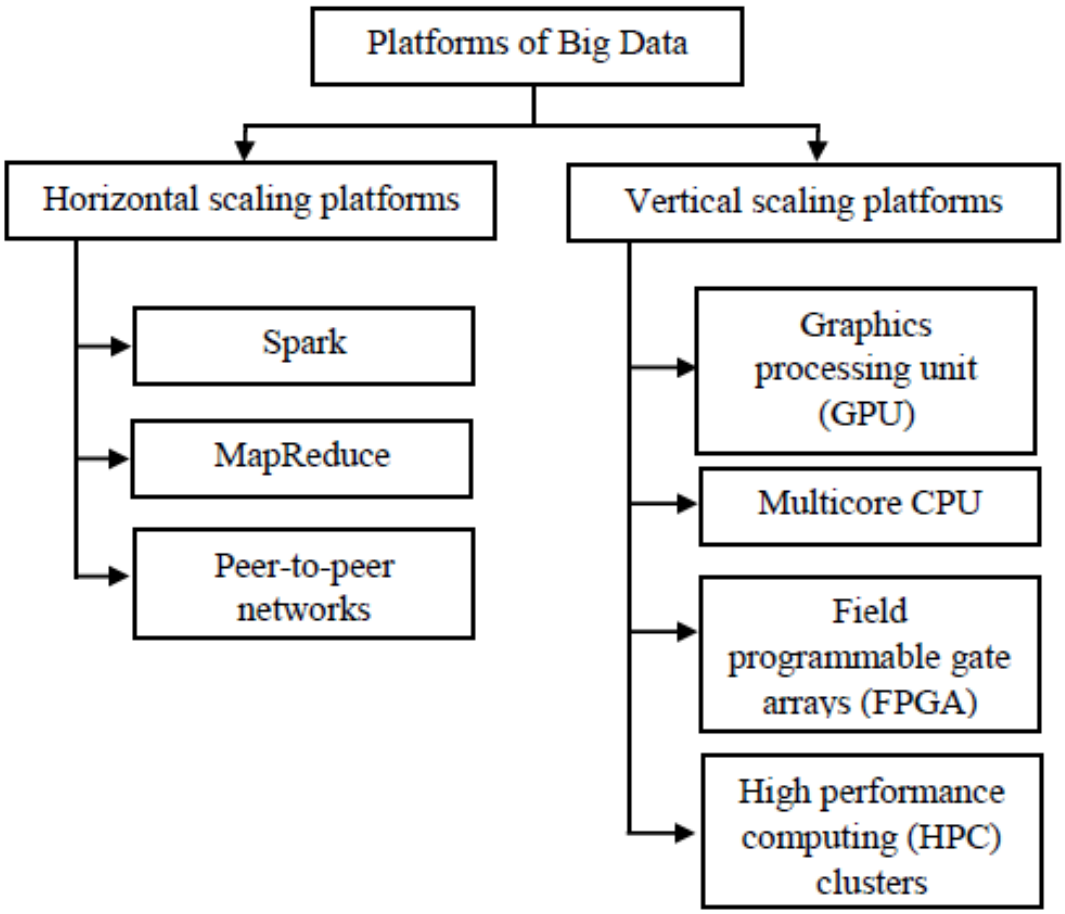

Figure 1.5: Classification of Big data platforms

1.2.1.5 Horizontal scaling platforms

MapReduce

MapReduce is a popular parallel programming model first introduced by Google at [40]. Designed to read, process, and write large amounts of data. This programming model consists of two main

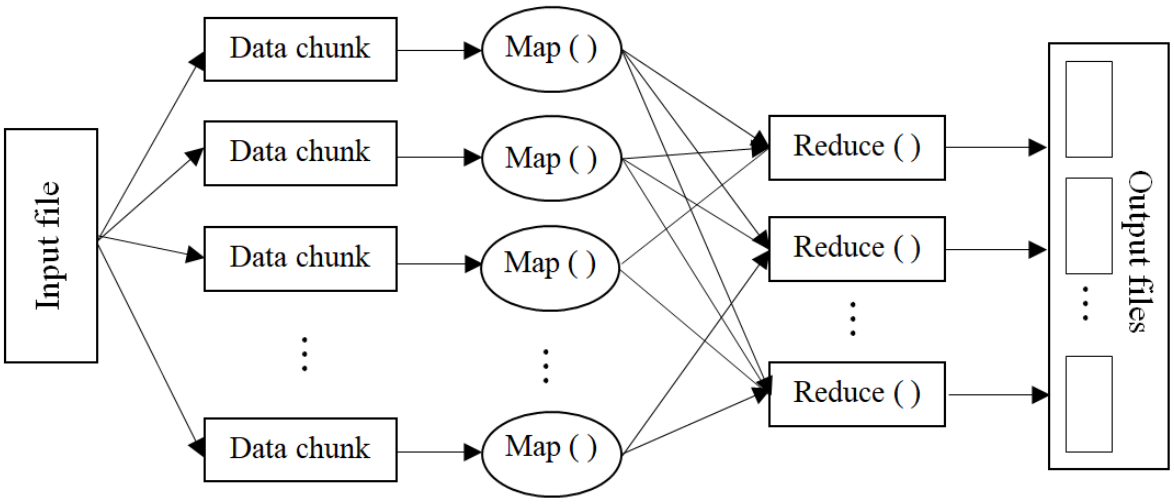

Figure 1.6: Flowchart of the MapReduce model

functions, the map function and the reduce function. A map function takes a logical data set (also called a data block) as input and produces a set of intermediate key/value pairs. After the Map phase completes, the Reduce phase begins processing the intermediate key-value pairs generated by the previous phase. In fact, the reduce function takes as input a set of intermediate key-value pairs that share the same key. Then it merges all the values associated with the Enter key to produce a set of values associated with the corresponding keys. There are two other optional functions used to coordinate program execution. Partitioner and combiner functions. The partitioner function splits the intermediate key according to the number of reduced tasks or the number of output files specified by the developer. The Combiner function aims to summarize the intermediate results produced by each Map task to avoid potential repetition. In fact, it helps optimize data transfer over the network to reduce tasks. All of these features are developer programmable. Figure 1.6 shows a working flowchart of the MapReduce programming model [40].

Spark

Apache Spark is a Big Data processing framework conceived for data-intensive applications and runs on commodity clusters [138]. Unlike MapReduce, The Spark framework handles iterative tasks and queries on large data sets by loading only useful data sets into memory. In this way

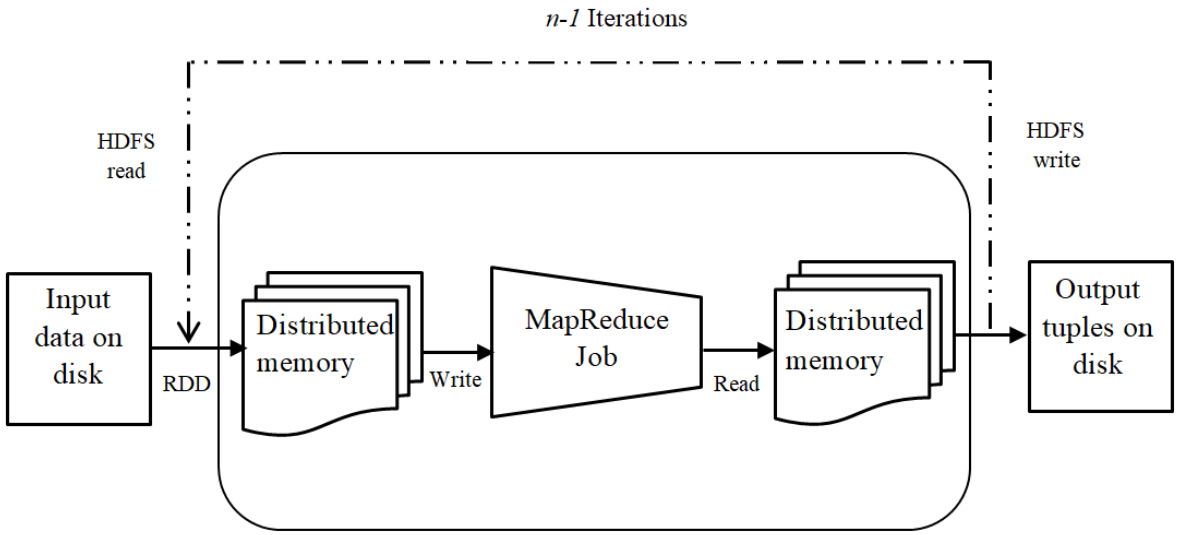

Figure 1.7: Flowchart of the Spark model

the execution time is significantly reduced.

Spark introduces three fundamental aspects: Resilient Distributed Datasets (RDDs), parallel operations, and shared variables. An RDD is a collection of objects that can be shared between many machines and restored if lost. It can also be stored in memory for reuse in multiple parallel MapReduce jobs. The second aspect is parallel operations that can be performed on RDDs. This includes Reduce, *Collect*, and *Foreach* operations. The final aspect consists of a broadcast variable and an accumulator. The Spark framework is flexible, easy to use, and requires no code abstraction. It also handles real-time data using the Spark streaming engine and stores partial results in memory using distributed workers. Additionally, Spark is efficient, maintaining MapReduce's fault tolerance and scalability, while delivering 10x the performance of Hadoop MapReduce frameworks for interactive machine learning workloads. However, Spark has some limitations, including the fact that it requires large resources and it is expensive in terms of memory. Figure 1.7 shows a working flowchart of the Apache Spark model [138].

Peer-to-peer networks

A peer-to-peer (P2P) network describes a distributed architecture that distributes tasks among peers. This definition of a P2P network includes any type of network architecture that does not require a server to control the transmission of information between contributors (peers). And

in such an architecture, peers make some of their resources available to other contributors in the same network [96]. P2P systems thus allow to benefit from valuable externalities, to reduce operating and sharing costs and, finally, to ensure anonymity. The most commonly used scheme for this platform is the Message Passing Interface (MPI). The basic principle of standard MPI is to provide the necessary abstractions to ensure peer-to-peer communication. It is also distinguished by its ability to keep processes alive while the system is running. Thus, there is no need to read the data from the diskette several times. This property makes MPI suitable for repetitive tasks.

The P2P network architecture has the ability to add nodes dynamically. As a result, it scales efficiently to larger sizes that may be required to handle applications with large amounts of data, as shown in Figure 1.8. This platform is known for its high resilience. This means that a local failure on one node will not affect the rest of the nodes. Another advantage is that the capacity of the P2P system increases as new nodes are added. However, adding new nodes to the system can slow down the transfer of data to connected users. P2P systems also suffer from security issues and must use high bandwidth.

1.2.1.6 Vertical scaling platforms

Graphics Processing Unit

A graphics processing unit (GPU) is a single-chip processor originally intended to handle 2D and 3D computations. As input, it takes geometry primitives in 3D format from the CPU. Then, convert them from individual vertices to pixels that are shaded and mapped to the screen. To produce the final image, the processed pixels are combined to form an output intended for visualization on a display device. These stages form a so-called graphics pipeline, as shown in figure 1.9 [100]. GPUs follow the Single Program Multiple Data (SPMD) programming model and Single Instruction, Multiple Data (SIMD) parallel architecture.

Today, modern GPUs are gaining increasing attention due to their massively parallel processing architecture that accelerates the performance of applications requiring highly floating point computations. In fact, GPUs aren't just for graphics applications. They are also used to perform non-specialized computations that lead to general purpose graphics processing unit (GPGPU) computations. New parallel programming languages such as CUDA [99] and OpenCL [120] have

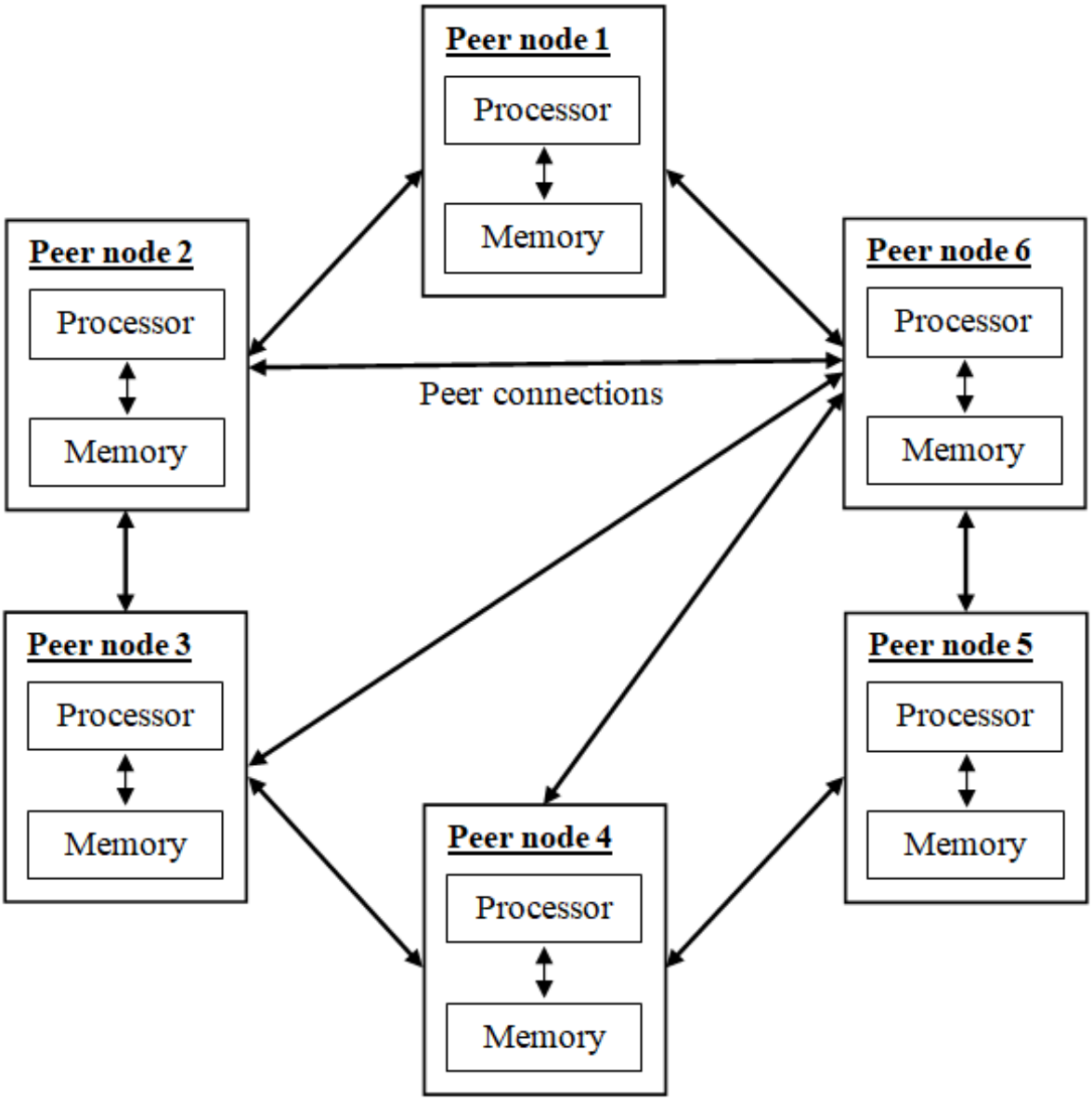

Figure 1.8: A typical architecture of the peer-to-peer network

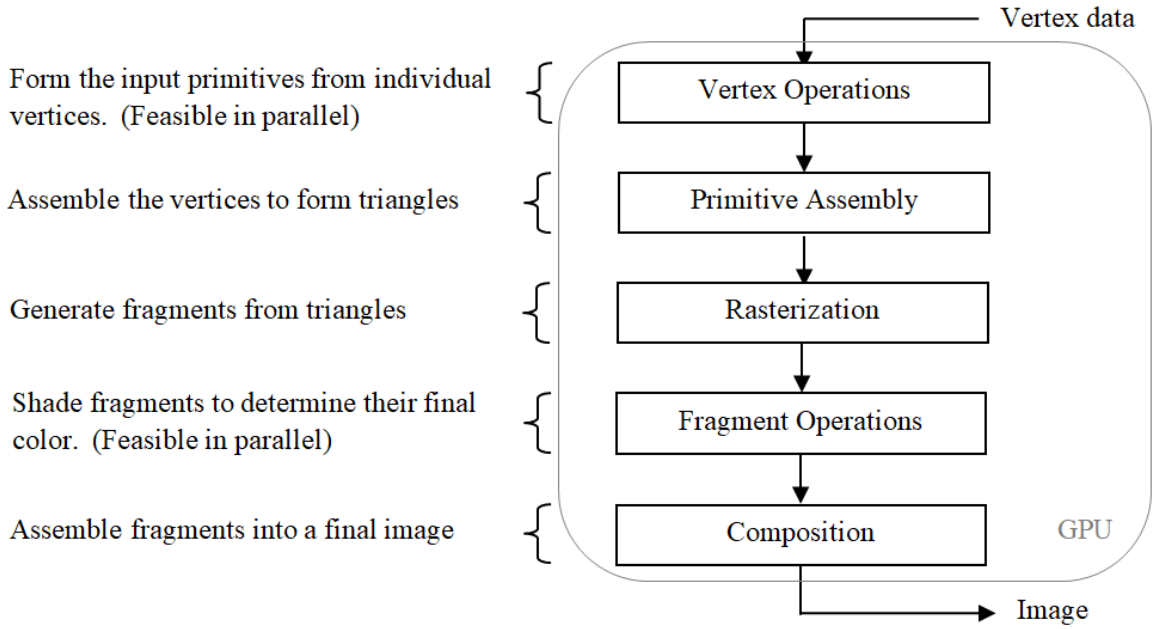

Figure 1.9: A typical structure of the graphics pipeline

emerged to take full advantage of GPU parallelism. These languages simplify and improve linear algebra performance on GPUs [100]. GPUs have demonstrated power and cost efficiency for compute-intensive and streaming storage problems. However, their biggest drawback is their limited storage capacity, which often requires complex storage management.

Multi-core CPU

A multicore platform is a processor that has multiple cores integrated into a single chip. In general, there are three generally accepted architectures for multicore CPUs [5]. The first architecture shares an on-chip cache between execution units, while the second provides a dedicated cache for each execution core. The third architecture uses a hybrid approach that divides the cache into two tiers: one dedicated to a specific execution core and one shared by all execution cores. Figure 1.10 shows a typical architecture for a multi-core platform.

To take advantage of the parallelism offered by multi-core platforms, we must consider dividing the work across all available processors [50]. The multithreading model is a common way to enable parallel execution on multicore platforms. This is achieved by dividing the work into separate execution units that can run concurrently on different processors. Multi-core platform

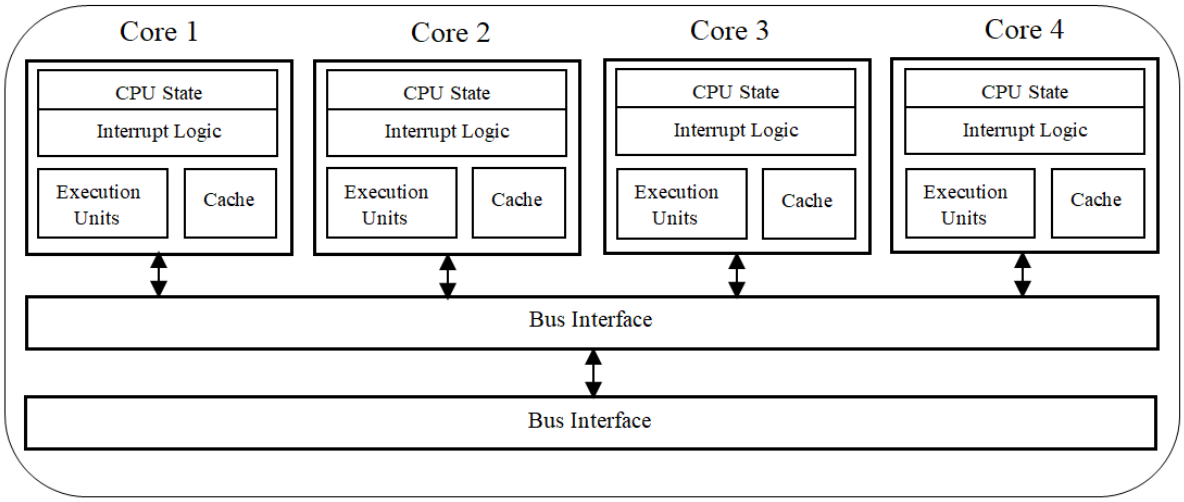

Figure 1.10: A typical architecture of the multi-core platform

provides high performance and low heat. These are useful for applications with high concurrency that can benefit from all available cores. Otherwise, multicore platforms are not a viable option. This requires considerable effort to parallelize the application as much as possible, which is not always possible.

Field Programmable Gate Arrays

A Field Programmable Gate Arrays (FPGA) is an integrated circuit device. It is composed of an array of programmable logic blocks, as well as a hierarchy of reconfigurable interconnects and I/O blocks [24]. Different architectures of logic blocks can be conceived to form a complex circuit. Such circuits contain several other sub-circuits and have more than one output. These blocks are interconnected via reconfigurable interconnects, which consists of wire segments and programmable switches. Like the logic blocks, the structure of programmable switches can be conceived in different ways. The configurations of all components of the FPGA are described using a hardware description language (HDL).

The main advantages of FPGAs lie in the reduced costs of prototypes they offer, in addition to their expandability and flexibility. In fact, flexibility is regarded as both an advantage and a drawback since it makes the FPGAs larger, slower, and more power-consuming [46].

The FPGAs are omnipresent in various applications and can figure out any computational

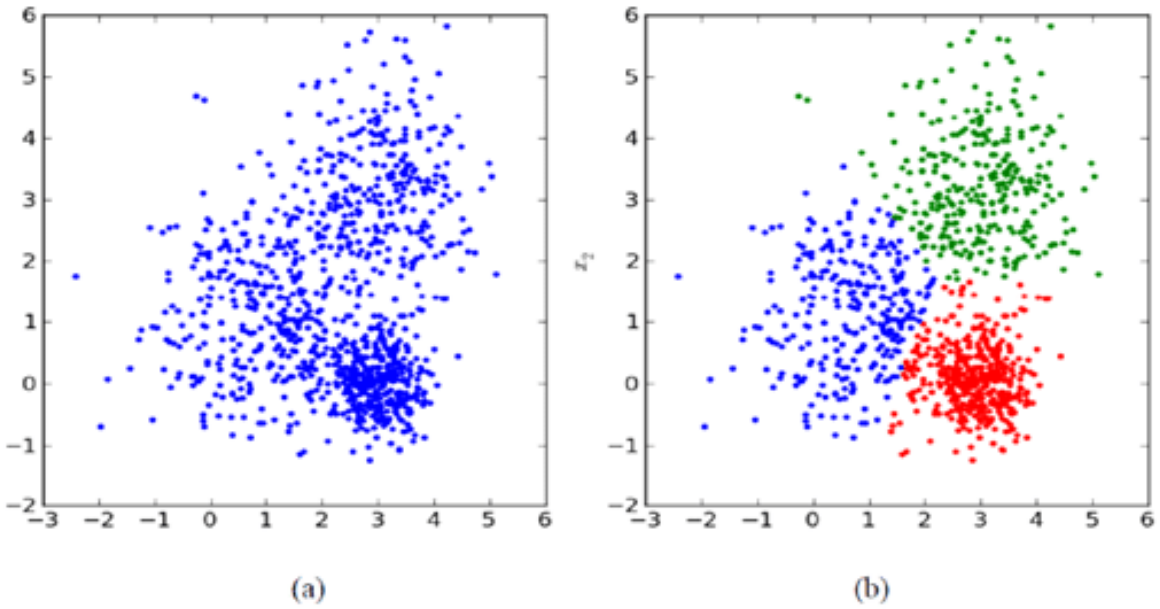

Figure 1.11: Example of clustering a data set (a) into three clusters(b)[48]

problem, especially the applications that require exploiting the parallelism available on FPGAs.

1.2.2 Cluster Analysis

1.2.2.1 Definition

Cluster analysis is a data mining task that aims to partition a set of data objects into clusters. Each cluster gathers data objects that are similar to each other, in such a way that two data objects belonging to the same cluster have a high similarity, while two objects belonging to two different clusters are completely dissimilar. A clustering algorithm is defined by several characteristics including the similarity measure, the input parameters, its stability, and the shape of the obtained clusters. Therefore, the clustering result depends on the applied clustering method, and therefore it can be different when using the same data set [57]. Figure 1.11 shows an example of clustering a dataset into three clusters.

1.2.2.2 Cluster Analysis Requirements

As the development of technology and the digitalization of the majority of services have increased, data generation has grown remarkably and continually, and data clustering has undergone intense growth to process heterogeneous and extensive data efficiently. In data mining, a clustering algorithm has several requirements. Indeed, a clustering algorithm must be highly scalable to handle large databases. It must also be able to handle different types of attributes, including binary, nominal (categorical), ordinal, and mixtures of these data types. In addition, the requirements of clustering algorithms also include, the capability to find clusters of arbitrary shapes such as spherical clusters, which can be achieved by using the Manhattan Euclidean distance, determination of input parameters which depend on the application domain, possibility to handle noisy data, incremental clustering and input order independence, capacity to cluster high-dimensional data, and interpretation and usability [57].
In addition, clustering methods are classified according to several orthogonal aspects, in particular, partitioning criteria that can represent clusters according to two possibilities. The first is to define clusters at the same conceptual level without any hierarchy among clusters. The second possibility is to form clusters at various semantic levels (with a hierarchy). A further criterion for clustering is the separation of clusters. In this way, each data object can be classified into a single cluster. In this case, the obtained clusters are mutually exclusive. The second possibility is to classify the same data object into several clusters. The similarity measure is also an important criterion for the conception of clustering methods. Thus, the similarity between two data objects is determined by the distance between them in a specific space, which can be a Euclidean or vector space, or any other space [57].

1.2.2.3 Basic Clustering Methods

There are several clustering methods that vary according to several aspects, including separation criteria, cluster segregation, similarity measure, and several other criteria. Thus, the main categories of clustering methods are partitioning methods, hierarchical methods, density-based methods, and grid-based methods [57]. Figure 1.12 illustrates the main categories of clustering methods.

Partitioning Methods

The purpose of partitioning methods is to construct mutually exclusive clusters having a spherical shape based on the distance between the data objects. Accordingly, a partitioning method proceeds by creating k initial partitions. Each partition has a center that represents, for example, the mean or the median. Then the partitioning is improved by using a specific iterative technique that aims at moving the data objects from one cluster to another by updating the cluster representative. In general, these methods are effective for small and medium data sets.

Hierarchical Methods

In hierarchical methods, data objects are decomposed on several levels, either in a bottom-up (agglomerative) or top-down (divisible) manner. The quality of the hierarchical agglomerative can be improved by several techniques including, analyzing the links between data objects in each hierarchical division, performing micro-ranking, or using iterative relocation. A hierarchical method cannot correct erroneous splits or merges.

Density-based Methods

Regarding density-based methods, data objects are grouped into spatially dense regions. Thus, clusters are separated by low density regions. Each data object must have a minimum number of data objects in its neighborhood. This clustering method is designed to filter out outliers. The most famous density-based methods are DBSCAN, DENCLUE, and OPTICS.

Grid-based Methods

A grid-based method performs clustering by quantizing the space into a finite number of data cells based on a multi-resolution grid data structure. The major advantage of these methods is that the processing time is fast and depends on the size of the grid. STING and CLIQUE are examples of grid-based methods.

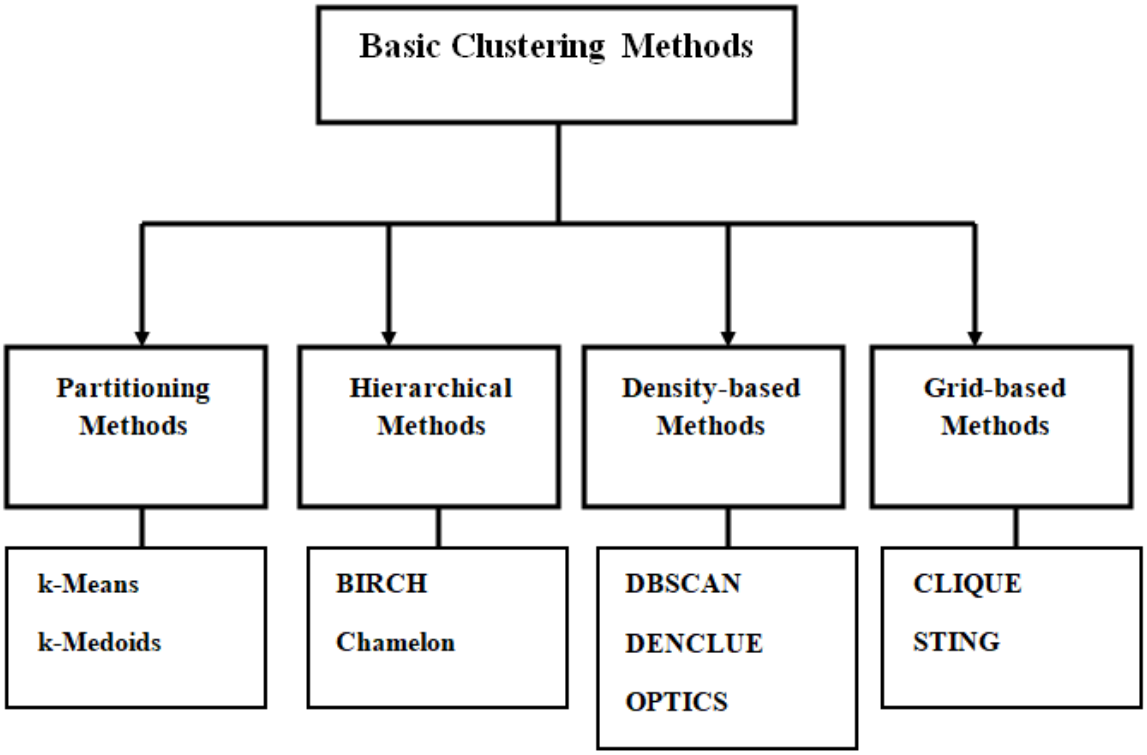

Figure 1.12: Basic Clustering Methods

1.2.2.4 Cluster Analysis Applications

Cluster analysis has recently become a very active topic in data mining research due to the vast amount of data collected in databases. The applications of clustering are multiple, such as business intelligence, which uses clustering as a way to organize customers who share the same interests and characteristics into the same group, which simplifies decision making and improves customer relations. Clustering is also used for image pattern recognition by building clusters or subclasses for handwriting recognition systems. In the field of web search, the results of a specific web search can be grouped into clusters to facilitate access to the results obtained. Another use of clustering manifests in outlier detection that has several applications, including E-Commerce Credit Card Fraud Detection and Criminal Activity Monitoring. Clustering can be used even in biology, medicine, and security and many other domains [57].

1.3 Motivation

The motivation of this thesis is related to the need to improve and adapt artificial intelligence techniques for efficient and beneficial handling of Big Data. Indeed, artificial intelligence and Big Data are inextricably linked to the point that it would be useless to have Big Data without being able to process it with powerful artificial intelligence tools, allowing the elaboration of new data types with complex structures.In addition, artificial intelligence tools require large amounts of data to learn effectively and therefore analyze this Big data and make thoughtful decisions. Artificial intelligence is able to recognize different types of data from different sources, find links between data sets, detect flaws and misinformation, as well as accelerate data preparation tasks. AI, therefore, acts on all phases of the Big Data cycle, starting with data preparation until data mining. Clustering is a widely used technique to categorize data sets in a homogeneous way to extract relevant knowledge according to the application domain. This data-mining technique relies on multiple algorithms that are distinguished according to the types of clustering, including, partitioning-based, hierarchical-based, density-based-, grid-based-, and model-based clustering. In this thesis, the partitioning-based clustering is exploited based on two main approaches. The first approach is the Relational Analysis approach, allowing to perform clustering without fixing the number of clusters or specifying complex input parameters. These advantages have been exploited in our research to improve the performance of the $Transitive$ heuristic [117] dedicated to the clustering of large categorical data. This idea was also used for the development of new algorithms for quantitative data processing, which are also able to solve the major problem of k-means, related to the determination of the number of clusters k and the determination of initial centers. The second approach is parallel computing, which allows the processing of Big Data by parallelizing the tasks of the clustering algorithm using the Spark framework [138], which has been used for its multiple advantages. Among them, the execution of queries in memory, the processing of iterative tasks, suitable for real-time processing, as well as its overtaking of the Hadoop MapReduce tool for interactive tasks of machine learning. The issue that arises is how to benefit from these approaches to develop new efficient algorithms for Big Data processing. To address this issue, the main contributions to this thesis are presented in the following section.

1.4 Thesis Contributions

The fundamental contributions of this thesis are as follows:

1- An overview of the latest parallel clustering algorithms categorized according to the Big Data computing platforms was proposed, by including a comparison based on some common criteria of clustering validation in the Big Data context.

2- Improving the quality of the $Transitive$ heuristic by taking advantage of the Relational Analysis approach. The enhancements suggested concern some weaknesses of the original heuristic, namely, the calculation of the representative and the manner in which each data object is processed.

3- Combining the E-Transitive heuristic adapted to the process of quantitative data and the k-means algorithm to automatically determine the number of clusters and the initial cluster centers. In addition, this contribution aims to provide different variants of the parameter-free clustering algorithm (PFK-means) focusing on the initialization process concerning two principal approaches: Overlapping clustering and Hard clustering.

4- Developping an efficient clustering algorithm for Big Data based on the main steps of the PFK-means algorithm. The parallel implementation is performed using the Spark Framework, Haddop, and the machine lerning based model.

1.5 International Publications

- International Journals:

  1. [37] **Dafir, Zineb**, Yasmine Lamari, and Said Chah Slaoui. "A survey on parallel clustering algorithms for big data." Artificial Intelligence Review 54.4 (2021): 2411-2443.

  2. [115] Slaoui, Said, and **Zineb Dafir**. "A Parameter-free Clustering Algorithm based K-means." International Journal of Advanced Computer Science and Applications 12.3

(2021).

3. [38]**Dafir, Zineb**, and Said Slaoui. "An efficient parallel algorithm for clustering Big Data based on the Spark framework" (2022).

- International Conferences:

1. [116] Slaoui, Said Chah, **Zineb Dafir**, and Yasmine Lamari. "E-transitive: an enhanced version of the transitive heuristic for clustering categorical data." Procedia Computer Science 127 (2018): 26-34.

2. [39]**Dafir, Zineb**, and Said Slaoui. "Achievements of artificial intelligence in the past and during the COVID-19 era to tackle deadly diseases" (2022).

1.6 Thesis Organization

- Chapter 1: The first chapter of this thesis includes the general context and problematic issues. Then, it describes the basic concepts of this research. After that, it presents the main contributions and international publications. And at last, it provides the thesis organization.

- Chapter 2: The second chapter provides an overview of the latest clustering algorithms for Big Data, categorized according to the Big Data platforms. In addition, it gives a thorough comparison of the reviewed clustering algorithms based on several Big Data criteria.

- Chapter 3: In this chapter, an enhanced version of the $Transitive$ heuristic for clustering large categorical data is presented. The proposed heuristic aims to improve the quality of the $Transitive$ by modifying the way of the calculation of the representative of a cluster and the manner to process data objects based on the Relational Analysis approach.

- Chapter 4: This chapter explains the principal steps of the prameter-free clustering algorithm based on k-means. It also describes the different variants of this new algorithm concerning overlapping clustering and hard clustering.
- Chapter 5: In this chapter, an efficient clustering algorithm for Big Data is presented. This new algorithm aims to parallelize the main steps of the parameter-free clustering algorithm based on k-means, using the Spark RDD's functions and the machine learning-based model to process a huge amount of data.
- Chapter 6: This chapter provides a general conclusion and summarizes the contributions suggested in this thesis. It also presents the prospects and future directions of this research field.

CHAPTER 2

STATE OF THE ART

This chapter provides an overview of recently developed parallel methods for clustering big data and current research trends in this related field. Additionally, the latest and most relevant parallel algorithms for big data clustering were reviewed and analyzed to explore different types of clustering, including: B. Partitioning-based clustering, density-based clustering, biologically inspired methods, and many other clustering techniques. All validated algorithms were analyzed according to the strategy adopted to ensure concurrency. Figure 2.1 shows the most important work examined by clustering type and big data platform. This chapter contains the first contribution published in the international journal [37], entitled "Survey on Parallel Clustering Algorithms for Big Data".

2.1 Horizontal scaling platforms-based clustering algorithms

The horizontal scaling platforms considered in this chapter are MapReduce, Spark, and Peer-to-Peer networks. This section covers a selection of clustering algorithms that are implemented using this kind of platform.

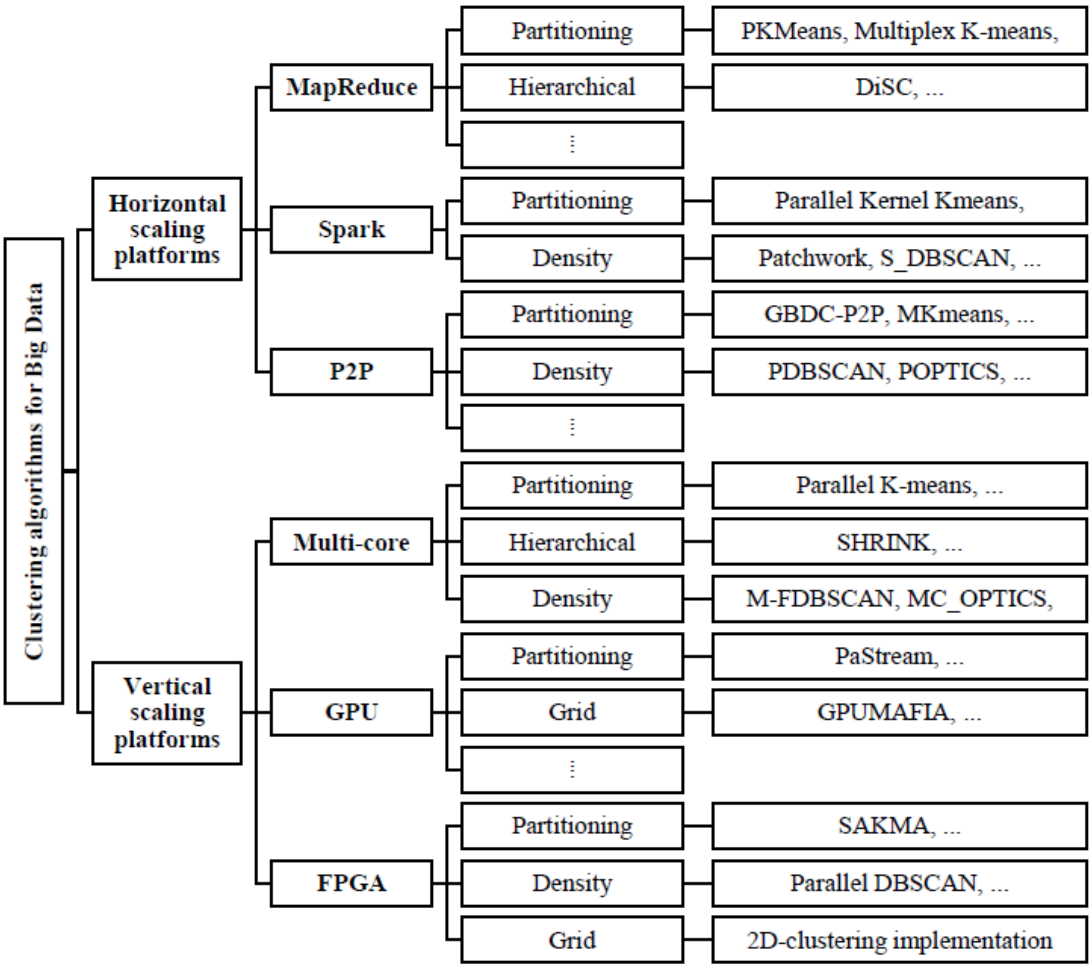

Figure 2.1: Parallel Clustering Algorithms for Big Data

2.1.1 Clustering algorithms using MapReduce

Partitioning-based clustering

The work proposed in [35] is a processing model in MapReduce that eliminates the iteration dependence of the k-means algorithm through a sampling technique. The main idea of this optimized k-means is to estimate the iterations using a sampling technique in order to obtain only some subsets from big datasets. Then, by processing these subsets, the sets of centers are constructed and used to cluster the original datasets. The proposed algorithm consists of three MapReduce jobs. The first job is responsible for sampling the original dataset. The second job performs the samples clustering step in mappers, and then the merging step in one

reducer in order to produce k final centers from the intermediates centers. For this purpose, the authors introduced two novel methods for merging: *weight-based merge clustering* (WMC) and *distribution-based merge clustering* (DMC). At last, the third job generates the Voronoi diagram using k points from the previous job, partition the original dataset, and then, obtain the final clustering result. The experiments on synthetic and real datasets show that the proposed method performs better compared with other parallel versions of the k-means algorithm.

In the same context, [80] suggested Multiplex k-means, a parallel execution of multiple k-means processes using MapReduce. In the proposed method, several processes are launched serially, then only the best result is considered. The execution of these processes is costly in terms of time and resources, unlike the proposed Mux k-means algorithm, which simultaneously runs multiple k-means using different centroid groups, and keeps the best solution at the end. The proposed algorithm involves four steps. It first runs the k-means processes using Map and Reduce operations. The Map operation calculates the distance between the current point and all existing centroids, while the Reduce operation is responsible for updating the centroids. After each iteration, the quality of the clustering result is evaluated based on the Total Within-Cluster Variation value (TWCV). The TWCV metric is the sum of the distances between each point and the centroids of each group. The more the value of TWCV is lower, the more the quality of clustering is higher. Then, in the second step, the groups with a lower value of TWCV are pruned. The third step consists of swapping similar centroids. Finally, the last step consists of generating new centroids using two methods: the *Random Search with a definite Scope* (RSDS) and the *Average of Dissimilar Group Pairs* (ADGP). This process is repeated until the centers become steady. The algorithm was implemented using the Hadoop MapReduce framework and tested with real-life datasets. The experiments show that the Mux-K-means can achieve better results than when using other serial versions of the k-means algorithm.

The work presented in [121] is also part of the partitioning-based clustering. It is a parallel method combining the information bottleneck (IB) theory with the centroid-based clustering. Their main contributions include the use of the IB theory-based hierarchy clustering to determine the centroid of each Map computational node, in addition to the use of an objective method in order to determine the number of clusters. The parallel IB theory is based on MapReduce. The

MapReduce job is designed with multiple Map tasks and a single Reduce task. In this method, the data are divided into partitions, and then each partition is treated independently and in parallel by a Map computational node. In each Map computational node, the IB theory-based clustering method is applied to each partition in order to obtain the sub-centroid. All sub-centroids are gathered in the Reduce task in order to create a new dataset. Then, the IB theory-based clustering method is once again applied to the new dataset in order to generate the initial centroid of the global dataset. Once the initial center is calculated, the parallel centroid clustering based on an iterative MapReduce model, called Twister [42], is applied. It is designed with multiple Map tasks and a single Reduce task, which returns its output to the Map tasks iteratively. The parallel centroid clustering works as follows. First, the initial sample dataset is partitioned and the initial centroids obtained previously are mapped to each computational node. In each Map computational node, the sub-centroids are recalculated with the centroid-based clustering method. All sub-centroids are gathered in the Reduce computational node and the global centroid is updated. Then, the new centroids are sent to the main computational node to be used in the next iteration. The process stops when a certain difference, which is measured with Kull-back divergence, is less than the fixed threshold value. In order to visualize the clustering results, the multidimensional scaling (MDS) has been applied, as a dimension reduction method, on the DNA data used in the experiments. The results show that the developed method is better than a typical parallel k-means implementation.

Hierarchical-based clustering

The work suggested in [68] is a MapReduce implementation of distributed single-linkage hierarchical clustering algorithm (DiSC), which constitutes two categories of MR jobs: Prim-MR and Kruskal-MR. The Prim-MR consists of applying a PrimMapper job one time, followed by a series of KruskalReducer jobs. In that respect, each mapper executes the local sequential Prim's algorithm in order to obtain a list of MST's edges, which is the input of the KruskalReducer. Once the mapper task is completed, the reducer combines the intermediate MST's based on the Kruskal's algorithm. The Kruskal-MR job runs the KruskalReducer task iteratively until the final MST solution is obtained. The proposed MapReduce implementation improves the efficiency

of the DiSC algorithm as it achieves a speed-up of up to 160× on 190 computer cores.

Density-based clustering

MR-DBSCAN [61] is a MapReduce-based implementation of the well-known DBSCAN algorithm. This work introduces a new cost-based data partitioning method in order to take into account the density of points. The proposed method is a 3-stage end-to-end solution. The first stage produces small partitions from the whole dataset according to the spatial proximity. The second stage gathers partitions independently. Finally, the third stage aggregates the produced partitions in the previous stage in order to form the final result. This latter is achieved in two steps: *build merge mapping* and *relabel data*. The build merge mapping step first determines all pairs of intersecting partitions, then computes the global clusters and builds a mapping from local to global clusters. And the relabel data step adjusts the intermediate results of local clustering by replacing local cluster ID's with global ones and determines the type of all points. The MR-DBSCAN was evaluated using two large real-life datasets. A set of experiments was carried out in order to study the performance changes when varying some sensitive parameters related to the DBSCAN algorithm, such as *MinPts*, ε and partitioning methods. The results of the experiments confirm the efficiency and scalability of the proposed method.

Spectral-based clustering

A parallel method for spectral clustering using MapReduce was proposed in [69]. The spectral clustering is based on the evaluation of the sparse matrix eigenvalue. The calculation of the similarity matrix and other parameters used in this algorithm are expensive, hence the need to use MapReduce in order to reduce the computation time. The proposed parallelization of the spectral clustering consists of three steps. The first step calculates the similarity matrices, which are simplified by the use of the Map and Reduce operations. The second step calculates the k minimum feature vectors using the Lanczos algorithm [77]. Finally, the third step parallelizes the k-means algorithm. The parallelization of k-means is divided into two fundamental steps. The Map function calculates the nearest centroid for each point. The combiner function partially merges samples with the same centroid. Then, it sends the result to the Reduce function as

key-value pairs, where the key is the centroid and the value is the list of points having the same centroid. Finally, the Reduce function collects the points with the same centroid and updates the centroids values by calculating the average of each set of points assigned to the centroids. The process reiterates until stable centroids are obtained. The experiments were performed using the classic dataset using several properties (correctness validation, a test of speed-up ratio, analysis of scalability). The results show that the parallel spectral algorithm is efficient for processing large datasets.

Nature-inspired clustering

MR-ABC [16] is a MapReduce-based implementation of the artificial bee colony (ABC) for large-scale data clustering. The proposed method optimizes the assignment of large data to clusters through the minimization of the sum of squared Euclidean distance. This method involves two main operations: updating the centroids of clusters and evaluating the fitness. First, the proposed method generates initial solutions. Then, it updates the new centroid values for the employed bees. After that, the fitness is calculated and evaluated based on the sum of the squared Euclidean distance. Since this task is time-consuming, the author used the MapReduce model in order to calculate the fitness value. Then, each onlooker bee selects the centroid values that produce higher fitness from employed bees and updates them. This process reiterates until the number of iterations reaches a threshold value. In order to evaluate the MR-ABC algorithm, experiments were conducted on synthetic and real-life datasets. The results obtained show that the proposed method outperforms the PKMeans [146] and the parallel K-PSO [129] algorithms in terms of quality. Besides, the running time and speed-up results of MR-ABC with 10 Hadoop cluster nodes demonstrated that the MR-ABC can process large amounts of data in a reasonable time.

Another work related to the nature-inspired clustering technique was presented in [136]. The proposed method is a parallel implementation of the Ant Colony Clustering method based on semantic content using MapReduce (MBSC). The MBSC algorithm involves a single MapReduce job. The Map task splits the data records into data chunks. Then, it defines the key-value pairs as the length of the traversal path without dropping records and the set of the nodes traversed. In each step, the Map function reads the pheromone value and calculates the swarm similarity,

in order to decide either to drop or to select a record. In this way, the data records with the same similarity are gathered in the same cluster. The reduce task collects solutions from all data chunks given by ants and then update the pheromone value for the next iteration. The process is repeated until the final result is reached. The comparison shows that the MBSC algorithm is more efficient than k-means in terms of time when considering different MSBC parameters.

2.1.2 Clustering algorithms using Spark

Partitioning-based clustering

A Parallel Kernel k-means has been introduced in [124]. The proposed method, which expands a previous work called the Trimmed Kernel k-Means [123], handles the clustering of large datasets using the Spark framework. The proposed method begins by sub-sampling the data to perform the clustering of large datasets. Then, it calculates the Kernel matrix iteratively using the MapReduce programming model and writes it to the disk for future use. Thereafter, it reads the Kernel matrix from the disk in order to compute the Kernel matrix trimming. The calculation of this matrix consists of two main steps. The first step finds the winning cardinality of each node and trimmed rows accordingly using the Map function. The second step sums up in the same vector the votes of the cardinality of each cluster using the Reduce function. The Map and Reduce functions are used again to remove the winning votes. This process reiterates until finding the cardinality of all nodes. The next step applies the Kernel k-means to adjacency lists given by the calculated Kernel matrix trimming. The process of Kernel k-means proceeds as follows: first, it associates each sample to a cluster randomly and then calculates the partial sum of the entries retrieved from the adjacency list using a mapping function. The reduce function calculates total sums from every cluster. Thereafter, the map function performs the distance computations in order to define the new cluster assignment for each node. The last distributed operation is the nearest neighbour assignment. The performance of the proposed method was evaluated using the Youtube Faces dataset, and compared with the approximate Kernel k-means and the Trimmed Kernel k-means algorithms. The clustering results show that the Parallel Kernel k-means algorithm is more efficient than the Trimmed Kernel k-means algorithm, and yields results close to those given by the approximate Kernel k-means algorithm.

Another work using Spark was suggested in [91], which is a framework for clustering and classification large amounts of data. The k-means and ID3 algorithms were studied and implemented using the proposed framework. The proposed framework is the parallel execution of multiple round-trips performed by the master node and several workers. Each round-trip consists of three main phases: Local Pre-processing, Global Distributed Processing, Local Post-processing. In the first phase, the master node requests to the workers to compute intermediate results from the stored data as part of the Map function. In the second phase, the workers swap the results obtained, and then collect the global information. In the last phase, each worker completes its own computations based on the intermediate results obtained from the previous phase. After having completed these phases, the master node runs a new round-trip. The data, which is swapped between the master node and the workers or between workers, operates in three modes: No-restriction mode, Partially Restricted Data Exchange Mode and Strict Restricted Data Exchange Mode. Additionally, the data processed by the workers can be executed differently according to the algorithm used in the proposed framework. The results show that the algorithms tested in the proposed framework were efficient in terms of time and scalability. Moreover, the k-means algorithm, which is implemented using the proposed framework, exceeds 31 % of the quality of results provided by the k-means algorithm using Spark.

There are several other works related to the implementation of k-means using Spark, such as [128], [140], and [113].

A parallel k-means algorithm using Spark was conceived in [128]. The proposed design can select the appropriate variant of the k-means algorithm and the distance function to use. The algorithm reads data from the HDFS and writes it in RDDs blocks using two different strategies. In the first strategy, each data instance is loaded directly into the RDD block in the form of a set of dense vectors, while in the second strategy it is represented by a set of sparse vectors. The next step computes the distance between every two sparse vectors in parallel. Finally, the last step performs two different methods for updating the centroids according to the clustering type used (crisp clustering or fuzzy clustering). For the crisp clustering, the assigning and the distance calculation steps are performed as part of the Map function, whereas the centroids updating step is performed as part of the Reduce function. This latter consists of collecting the instances with

the same centroid, and then compute their average value. For the fuzzy clustering, the centroids updating step is performed using a predefined equation.

Another work related to the k-means algorithm is proposed by [140], which is called the Parallel Overlapping k-means method (POKM). It was designed to perform non-disjoint partitioning of large-scale data. The proposed work is based on a geometrical method, called Overlapping k-means (OKM), which consists of two steps: assigning each instance to multiple clusters and updating centroids. These steps are repeated until reaching the convergence criterion. Following the same manner, the Spark-based version is based on two steps: the data assignment and prototypes calculation steps. Ones the input data is partitioned, the first step would be to apply a Map function for each block. It also assigns the instances to multiple clusters in order to parallelize the assigning procedure locally. Then, another function operates on the global clusters, which are produced previously, by sorting the prototypes of clusters according to their proximity to the processed instance. This process assigns the clusters from the nearest to the farthest as long as a predefined criterion decreases. The Prototypes calculation step is performed by Map and Reduce tasks. The Map task performs local computations for each cluster and sends as output a set of key-value pairs, where the key is the index of the prototype cluster and value is a couple of computed parameters. The Reduce task updates the prototype clusters and returns the final ones.

In the same context, [113] presented a method based on the classical k-means algorithm to process big data streams using the Spark framework. The proposed algorithm consists of four main processes. The first process consists of generating initial clusters using a large value of k. The second process applies the k-means algorithm to the k clusters. The next process consists of merging the centroids that are spaced from each other by less than a certain threshold value. The last process returns the final clusters by merging the previously obtained clusters. This algorithm is characterized by its free over-resolution.

The reviewed methods demonstrated their efficiency in terms of scalability and quality of clustering large data.

SRSIO-FCM [22] belongs also to the partitioning-based clustering family. It is a scalable random sampling and an iterative optimization fuzzy c-means algorithm. The proposed method divides randomly the dataset into many subsets. Then, it generates the centers of clusters

randomly in order to cluster the first subset. The cluster centers and the membership information of the first subset are calculated, and then they are given as input to cluster the second subset. In turn, the cluster centers, and membership information of the second subset are calculated. After the first two iterations that present a particular case, the membership information of all the processed subsets is combined in order to compute the new cluster centers, which will be given as input to cluster the next subset. The following subsets are clustered in the same manner. The authors proposed a parallel implementation of the LFCM algorithm [60] using Spark, which they called the SLFCM algorithm (Scalable Literal Fuzzy c-Means), in order to handle the parallel computation of membership information and centers of clusters. It performs Map and ReduceByKey operations. The Map phase is responsible for the membership degree calculation of a data point with respect to each center. And, the ReduceByKey phase is responsible for updating the values of the center of the cluster based on the output of the Map phase. The performance of the SRSIO-FCM algorithm was compared with the Spark based-implementations of two well-known fuzzy clustering algorithms. The experiments on some big datasets demonstrated that the proposed method achieves almost the same quality of clustering results in less time.

Density-based clustering

RP-DBSCAN [118], Random Partitionning-DBSCAN is a parallel DBSCAN algorithm based on the random split strategy. The parallelization of the proposed algorithm was performed using the MapReduce model and implemented with the Apache Spark API. In that respect, the RP-DBSCAN algorithm consists of three major phases: Data partitioning, cell graph construction, and cell graph merging. The first phase partitions the initial dataset using pseudo random partitioning, then constructs the two-level cell dictionary and sends both a partition and the two-level cell dictionary to every worker. The second phase aims to find the directly density-reachable relationships from each partition as a first step, and then gather the individual relationships at the cell level to obtain a cell graph. The last phase consists of merging the cell graphs collected from each worker in order to attain global clustering. The RP-DBSCAN algorithm was implemented using Spark, on various real-world datasets, and compared with five popular Spark-based implementations. The results obtained demonstrate the efficiency of the

developed algorithm in terms of accuracy and scalability. In addition, the RP-DBSCAN algorithm outperformed these sparks-based implementations of up to 180×.

Remaining within the context of the density-based clustering, the work presented in [84] is a parallel implementation of the density peaks clustering algorithm [104] using the Spark's API for graphs computation, called GraphX. The authors noticed that the original method needs to calculate the distances between all pairs of data points which results in high computational cost. In order to overcome this issue, they have proposed a parallel version of the density peaks clustering algorithm using the Spark RDD model. The process starts by initializing the threshold values of the local density and the distance from points of higher density and then generating a graph. The graph construction requires, first, importing separately vertex and edge data stored in the HDFS to vertex RDD and edge RDD. Then, computing the distance and updating its value on each edge. After that, the vertex RDD and edge RDD need to be combined in order to form a Graph. The next step consists of computing the truncated distance, the local density, and then the distance from points of higher density. The last step performs the clustering of data points by selecting cluster centers, isolated points, and then performs the classification based on the thresholds fixed previously. The experimental results demonstrated that the proposed implementation can be up to 10× faster than the density peaks clustering algorithm when implemented using MapReduce.

Patchwork [52] is a distributed density clustering algorithm to analyze very large datasets. It was designed using the MapReduce model to parallelize the calculations, and implemented with Apache Spark. The Patchwork algorithm is a part of density clustering algorithms that are known for their linear computational complexity and near-linear horizontal scalability. It consists of two fundamental steps. The first step consists of dividing the multidimensional feature space into a grid in order to determine dense regions and then finding the ID of the cell (hypercube in D-dimensional feature space) for each point using the Map function. After that, the Reduce function collects the tuples with the same cell ID and constructs collections of cells with their density. The second step consists of creating clusters using the collections of cells and sorting them by decreasing the density until all cells are processed. An optional filter is used to help find clusters with enough cells. The proposed algorithm was compared with Spark MLLib k-means

and Spark DBSCAN using four Synthetic datasets. It is efficient and yields 40× better results than those produced by Spark MLLib k-means.

Remaining within the context of the density-based clustering methods, several studies have been conducted on the DBSCAN algorithm in order to propose parallel implementations based on Spark, such as [89], [56], and [33].

The S_DBSCAN algorithm, introduced in [89], starts by partitioning the raw data based on random samples. It produces partial clustering results by performing local DBSCAN algorithm in parallel. At this stage, the Map task generates partial clusters, while the ReduceByKey task saves each partial cluster as a new RDD to the HDFS, and then computes the centroid for each partial cluster. Finally, it merges the independent clustering results obtained in the previous steps based on the centroid.

The proposed implementation in [56] starts by reading data from the HDFS and then generates RDDs in order to transform them into data points. In the next step, multiple executors build independently partial clusters and sent them back to the driver. When all the partial clusters are collected, the merging process starts identifying the clusters that need to be merged based on a new concept introduced by the authors, called SEEDs. SEEDs are points that do not belong to the current partition and they are placed by executors to serve as markers, in order to identify master partial clusters and then merge them. The proposed method is distinguished by the use of a Java-based kd-tree implementation in order to optimize the complexity of the task of searching for the neighbours of points.

The RDD-DBSCAN algorithm, introduced in [33], consists of five steps. This algorithm starts the same way as the Apache Spark-based implementations of DBSCAN described above. It divides data and then performs the local DBSCAN algorithm on each partition. After that, it identifies clusters which span multiple partitions and generates global cluster identifiers. Finally, all points are relabeled using the newly-found, globally-unique identifiers.

The three methods were assessed based on their accuracy and speed up. They have proven their efficiency in processing large data. The algorithm, proposed in [56], can achieve a performance of 16× faster than the MapReduce implementation of the DBSCAN algorithm. The experiments of the RDD-DBSCAN algorithm demonstrated that the communication costs do not

significantly affect its performance.

2.1.3 Clustering algorithms using peer-to-peer networks

Partitioning-based clustering

Recently, a gossip-based distributed clustering algorithm for peer-to-peer unstructured networks (GBDC-P2P) [13] was introduced. The GBDC-P2P algorithm is based on K-medoids and k-means algorithms for extracting the representative data and discovering the final clustering results, in addition to the CYCLON algorithm [127] to ensure the interactions between peers. It starts by selecting M representatives in each peer among its internal data. In the next step, which concerns the gossip-based interactions, each peer sends the representatives to the neighbour peers. This process of interactions between peers is based on the CYCLON algorithm. Then, the summarization step summarizes the external data of peers once their memory is full. This step is performed only when necessary. The property of the data summarization is provided by the K-medoids algorithm. The adaptation of the GBDC-P2P algorithm to dynamic network conditions is ensured using an age variable for each external data of peers. At each round of gossip-based operations, all peers increment the age variables of their external data. This mechanism allows replacing the old external data of peers. Finally, an improved version of the k-means algorithm, called Persistent k-means, is proposed and it is performed in each peer in order to calculate the final clustering results. The clustering results obtained by the GBDC-P2P algorithm are very close to those obtained by the centralized k-means algorithm.

Remaining in the same category of clustering algorithms, MKmeans [144] is a parallel k-means clustering algorithm implemented using MPI. This algorithm follows the data parallelism strategy, which consists of partitioning evenly the data objects in all processes and replicating the cluster centroids. For each of these processes, K initial centroids are randomly selected, and then each data object is assigned to the closest cluster. When all objects are assigned, the new centroid of each cluster is then calculated. This process is repeated until a user-specified threshold is met. Then, at the end of each iteration, the new cluster centroids are generated according to the clustering results of all processes. Lastly, the final centroid set is generated by a simple merging algorithm, and the clustering results are output as the K centroids, the I/O time, and

the clustering time. On the experimental side, the MKmeans algorithm has shown that it is relatively stable and portable. However, the experiments focused only on the time performance aspect. Indeed, the experiments lack the clustering validation aspect.

Density-based clustering

An MPI-based implementation of the DBSCAN algorithm is presented in [107]. The authors designed an approach based on three phases, namely, the splitting, the DBSCAN execution, and the final clusters forming. Furthermore, they adopted the SIMD (Single Instruction Multiple Data) technique, which allows executing the same task by each computational node of the system on different data. The master node is responsible for splitting and assigning the data to the worker nodes, while the worker nodes perform independently the DBSCAN algorithm. Consequently, each worker node produces its own local clusters in the form of a collection of pairs, wherein the centroids and the radius of the clusters are saved. Then, each worker node sends its collection to its neighbour node in order to explore possible aggregations of clusters. This is achieved by computing the intersections of the circles, then examining these intersections. In the case where the circles do not intersect, there is no action since they form different clusters. This process is repeated until all information becomes on the final worker node, which will emit the clusters as a final result to the master node. The experiments were conducted on 33 computational nodes and they demonstrated that the proposed approach reduces the time complexity and yields identical results compared to the original sequential DBSCAN.

PDSDBSCAN is also a parallel DBSCAN algorithm [101] that employs the disjoint-set data structure and a tree-based bottom-up approach. The proposed algorithm starts by creating a singleton tree for each point of the dataset. Then it combines the trees belonging to the same cluster based on the disjoint-set data structure to form the clusters representing the initial dataset. This step allows high data parallelization since it breaks the inherent data access by merging the trees in an arbitrary order. The last step consists of producing a single tree for each obtained cluster. The algorithm was implemented in both shared and distributed memory. Accordingly, the distributed implementation is more relevant as it deals with data larger than the one handled by the shared implementation. The experiments conducted on high dimensional

datasets, using MPI architecture, have demonstrated that the proposed algorithm can reach a speed-up of up to 5765x using 8192 process cores.

OPTICS is also a well-known member of the density-based clustering family and has been the subject of several parallelization attempts. Examples of successful attempts include the POPTICS algorithm [102]. POPTICS was designed using the concept of Minimum Spanning Tree (MST). The idea of this algorithm is to compute a MST for each processor on its local dataset without incurring any communication, and then, to merge all the local MSTs in order to obtain a global one. These steps are performed completely in parallel. Besides that, a parallel disjoint-set data structure is used in order to extract the clusters directly from the global MST in a highly parallel and distributed way. Although the POPTICS algorithm was parallelized using both OpenMP and MPI to run on shared-memory and distributed memory systems, respectively, the MPI-based implementation is particularly interesting because it can deal with larger datasets. Indeed, the experiments illustrated that the MPI-based implementation of the POPTICS algorithm can reach a speed-up of 3,008x using 4,096 processors on high dimensional datasets containing up to a billion floating-point numbers. As will be shown later in this paper, POPTICS is the basis of another interesting algorithm for clustering trajectory Big Data using GPGPU [41].

Grid-based clustering

As an example of grid-based clustering algorithms, we can mention an MPI-based implementation [137] of the WaveCluster algorithm [109]. The WaveCluster algorithm transforms the original feature space by adopting the wavelet transform, thus forming a new space in which the dense regions must be sought. As a result, it produces sets of clusters at different resolutions and scales. In order to ensure the parallelism of the WaveCluster algorithm, the authors adopted the replicated approach [114], which consists in splitting the dataset over processors that execute nearly identical code segments of the algorithm. Therefore, processors need to exchange their local results and to check the correctness. Furthermore, they adopted the master/slave model with the SPMD (Single Process, Multiple Data) technique on a distributed-memory multiprocessor system. The MPI was used to manage the communication requirements between master and slave nodes. Experiments were conducted on a PC cluster of 8 compute nodes with 32 processors

at total and having fast Ethernets as underlying communication hardware. The clustering results demonstrated that the proposed method yields high speed-up and scales linearly.

Graph-based clustering

DIDIC [49] is a distributed heuristic for clustering a virtual P2P supercomputer. It consists of three phases, namely, the establishment of the initial situation, the elimination of global knowledge with suitable diffusive processes, and the determination clustering. In the first phase, if no initial configuration is provided, then a random configuration is considered. In the second phase, the diffusive clustering process is carried out. Two diffusion systems are used per cluster to represent the same load colour. The primary diffusion system exploits the property of diffusion and random walks in order to identify dense graph regions. While the secondary system sends a load of the system to the nodes belonging to clusters in order to accelerate the forming of large cluster-connected components. The last phase aims to flood areas with new clusters when the diffusion process shows a really strong desire for this by integrating the isolated nodes. The authors adopted the BSP (bulk synchronous parallel) model [125], which provides an abstract view of the technical structure and communication features of the hardware, in order to ensure the parallelism of the proposed method. The clustering results of the DIDIC heuristic were compared to results obtained with the MCL algorithm [43], which tend to be close and sometimes slightly better in favour of the proposed method.

Spectral-based clustering

p-PIC [135] is an MPI-based implementation of the power iteration clustering (PIC) algorithm [82]. The original algorithm produces clusters by embedding data points in a low-dimensional subspace derived from the similarity matrix. It consists of three main operations: the similarity matrix calculation and normalization, the iterative matrix-vector multiplication, and the clustering. The authors of the p-PIC algorithm noticed that the original method depends on the memory resources because it stores data and its associated similarity matrix in memory. Therefore, they suggested enhancing the two first operations involved in this method by exploring the parallelization strategies in order to reduce the computation and communication costs. Indeed, each processor

stores only two cases in memory at a time, in addition to this, the similarity matrix is split according to the strategy of row-wise block stripped matrix-vector multiplication and stored at the processors executed in parallel. The experimental results on a local cluster and Amazon EC2 cloud show that the p-PIC algorithm reaches almost a linear speed-up on all tested datasets.

2.2 Vertical scaling platforms-based clustering algorithms

In this section, we present the recent clustering methods based on vertical scaling platforms, namely the GPU, Multi-core CPU, and FPGA platforms.

2.2.1 Clustering algorithms using Multi-core CPU

Partitioning-based clustering

A parallel variant of the k-means algorithm using multiple CPU cores of a single machine was suggested in [54]. The proposed algorithm divides the datasets into chunks and then distributes them to the processing units. The processing units perform the clustering on the chunks of data in parallel. The proposed algorithm consists of two thread sets: Master thread and chunk-clustering threads. The master thread is responsible for reading datasets and organizing them into chunks that have a predefined size. After that, the master thread sends the chunks using the concept of the queue. The next step consists of performing the clustering by multiple threads using k-means++ and then storing the centroids obtained from each chunk in the global list of centroids. Once the clustering of all chunks is completed, the global list of centroids is loaded by the master thread in order to run the next step, which consists of clustering the centroids using k-means++ to produce the final centroids. The proposed algorithm was evaluated using a 12-cores machine, and compared with the k-means, k-means++, and stream-based algorithms. The results show that this algorithm can achieve near-linear scalability, and yields the same quality of results as k-means++. In addition, it runs much faster than stream-based algorithms.

Earlier, the work published in [75] suggests parallel implementations of the k-means and K-modes algorithms using a multi-core platform with transactional memory in order to process a large amount of data. The k-means and K-modes algorithms were implemented as follows:

The first step consists of distributing the initial datasets to the cores. Then, each data point is assigned to the nearest centroid. The next step consists of updating the centroids by k data threads. Both assigning data and updating centroids operations are performed in parallel using a software transactional memory (STM), named deferred-update STM. Each data point is processed by running the threads simultaneously. The data must be verified before sending it to the shared memory. If any change has been made to the data, then the transaction is sent. Otherwise, the transaction is rejected. The parallel multi-core k-means algorithm (McKmeans) was compared to the single-core k-means algorithm implemented in R, by simulating artificial data, using a dual quad-core computer. Then, it was compared to the network-based ParaKMeans tested on the web with the same datasets. These comparisons show the high performance of the proposed algorithm. Besides, the speed-up achieved by the proposed algorithm is 10× better compared to single-core implementations.

Hierarchical-based clustering

Concerning the hierarchical-based clustering, the proposed work is a new shared-memory algorithm for single linkage hierarchical clustering [62], called SHRINK. The parallelization process of the suggested algorithm consists of three main stages. The first one aims to partition the original dataset into overlapping subsets having approximately the same size. The second stage computes the hierarchical dendrogram based on the highly efficient SLINK algorithm [111]. The final stage merges together the dendograms by combining partial MSTs using Kruskal's algorithm in order to raise the height. The steps of the last stage are repeated until all dendrograms are combined and then obtain the single dendrogram that represent the solution of the original problem. The implementation of the algorithm was performed using OpenMP. The experiments conducted on both real and synthetic datasets of up to 250000 points yield a speed-up of 18-20 on 36 cores.

Density-based clustering

M-FDBSCAN [44] is a multi-core density-based uncertain data clustering algorithm. This algorithm is an adaptation of the fast density-based spatial clustering of applications with noise

(FDBSCAN) [76] to multi-core systems. The idea behind the original FDBSCAN algorithm is to integrate the fuzzy distance functions directly into the traditional DBSCAN algorithm in order to measure the similarity between fuzzy objects. The adaptation of this clustering algorithm to the multi-core platform resulted in the M-FDBSCAN algorithm, which has demonstrated a significant acceleration of processing. The newly introduced algorithm operates in three steps. First, it splits the 2-dimensional fuzzy data object dataset into subsets according to the number of available cores in the multi-core system. Then, it applies the FDBSCAN algorithm to each subset concurrently, which leads to the determination of the final cluster regions partially. Finally, the last step in this algorithm consists of merging the subsets pairs to get the final cluster regions. The M-FDBSCAN algorithm is implemented using OpenMP in C to achieve its parallelism. Experiments conducted on synthetic datasets demonstrated that the proposed algorithm scales linearly when increasing the number of cores and it outperforms the FDBSCAN algorithm when processing huge amounts of data.

In the same context, MC_OPTICS [53] is a parallel version of the OPTICS algorithm for shared memory multi-core systems. Since the most time-consuming parts of OPTICS are the neighbourhood queries and core distance computations, the authors of MC_OPTICS proposed to perform the neighbourhood and the core distance computations concurrently using a master-slave pattern with priority queue updates. Indeed, the proposed algorithm adopts both data and task parallelism strategies. The task parallelism strategy is achieved through the master-slave model where the master thread distributes the work to slave threads and the data parallelism is ensured by the slave threads. The master thread is in charge of the priority queue operations and the data distribution to local queues of slave threads. While the slave threads are in charge of the neighbourhood and the core distance computations, each for its subset of data, and independently of each other. MC_OPTICS produces the same output as the sequential OPTICS thanks to the priority queue, in which points are inserted, dequeued, and written into the cluster ordering list in the same order as in sequential OPTICS. Experiments on real-life and synthetic datasets, reaching 1.1 million data points, have demonstrated that MC_OPTICS yields scalable performance.

2.2.2 Clustering algorithms using Graphics Processing Unit

Partitioning-based clustering

Three parallel implementations of the k-means algorithm are suggested in [36] for the purposes of reducing the execution time using GPUs and CUDA. These implementations follow a hybrid approach, in which the k-means algorithm is performed partly in the host and partly in the device. Data transfer between host and device is efficiently carried out using coalescing memory accesses and pitch for matrices memory allocation. The proposed parallel approach calls the device at each iteration. This requires fixing some parameters and checking the dimension of the matrix of data points. Some dimensions need the data points to be divided into chunks and then process them one by one. Then, the host calculates the new centroids using the information of the new indices produced by the device, and finally, it checks the convergence. The authors observed that the k-means algorithm needs three data structures to store the set of data points, the centroids, and the indices that indicate the membership of each data point. In order to address this observation, they proposed a lighter data structure that can reduce the data transferring time. In their first implementation, they proposed to use a matrix to store the data and to update the results for the labelling stage on the CPU. While in the second implementation, they used two different data structures to store them. The third implementation, which is considered the most refined solution, adopts a parallel technique to compute the squared Euclidean distance. This refined implementation achieves a speed-up reaching 88× faster than the CPU version.

PaStream [63] is a parallel algorithm for clustering data streams based on NVIDIA GPUs and which make also use of the k-means algorithm. This single-pass algorithm discovers clusters with arbitrary shapes and detects outliers. The PaStream algorithm is implemented following a framework for clustering fast-evolving data streams, called CluStream [2]. The PaStream algorithm requires an initialization step, in which some micro-clusters are generated using the k-means algorithm. Then, it involves two phases: an on-line phase, in which data points are gathered into micro-clusters, and an off-line phase, in which micro-clusters are aggregated into macro-clusters. In the on-line phase, for each data point, three cases must be considered. In the first case, an existing micro-cluster absorbs the data point. In the second case, the data point

forms a new micro-cluster, while in the third case, two closest micro-clusters merge to form one. This decision is made according to the absorption rate. Furthermore, the authors have defined a merging factor, which determines when the close micro-clusters should be merged in order to optimize memory and time. In the off-line phase, the Euclidean distance between all centers is calculated, then, the decision graph is plotted based on the local distance and the local density for each micro-cluster. It should be noted that this algorithm operates on data groups and not on data points at its arrival. This helps reduce the time required to transfer data from CPU to GPU. The experiments on real and synthetic datasets show that the PaStream algorithm outperforms its version based on CPU in terms of time and quality of the produced result.

Density-based clustering

The work proposed in [94] is part of the density-based clustering. It focuses on the parallelization of the OPTICS algorithm using GPU and based on the data indexing strategy. The OPTICS algorithm creates an augmented ordering of the dataset representing its density-based clustering structure [9]. The proposed parallel implementation of the OPTICS algorithm consists of two phases: the graph construction and the OPTICS process. The proposed approach adopts a data representation in a graph form following the METIS data structure, which is presented in [71]. And it uses three vectors to store the vertices, the nodes in the adjacency list, and the distance. The process consists of four main steps, respectively, the vertices degree calculation, the adjacency index calculation, the adjacency lists assembly and sorting. These steps, in addition to the construction of data structure and the storage at the end of the adjacency lists of nodes, are each processed in parallel using GPU. The experiments show that the proposed approach reduces significantly the complexity of the OPTICS algorithm and makes it faster than the serial CPU-based version.

In the same context, authors of [41] have proposed a new trajectory clustering approach based on the POPTICS algorithm [102]. The POPTICS algorithm was described in a previous section. In order to adapt this algorithm to the trajectory data, the authors applied a spatiotemporal distance to measure the similarity between trajectories and an indexing approach based on the STR-tree structure to their proposed algorithm. This latter, which is called Tra-POPTICS, is based on

shared memory and involves three main steps. In the first step, each CPU thread processes a local disjointed subset of trajectory data. It finds out the neighbours of each trajectory. Then, it computes the core distance of each trajectory and then local Prim's minimum spanning tree (MST). The next step generates the global MST, and finally, the last step extracts the clusters from the global MST. As part of this study, a parallel version of the Tra-POPTOCS algorithm, called G-Tra-POPTICS has been designed using the Hyper-Q feature of Kelper GPU and massive GPU threads. The experiments on real trajectory dataset demonstrated that the Tra-POPTICS algorithm and its parallel version have reached the quality of the clustering result produced by a variant of OPTICS algorithm called T-OPTICS [98]. In addition, the GPU-based version outperforms the Tra-POPTICS algorithm regarding computational time.

Another work related to the density-based clustering is presented in [8]. It is a GPU-based implementation of the DBSCAN algorithm, called G-DBSCAN, which uses a simple graph-based data indexing technique. The proposed approach consists of two main steps, namely, the graph construction and the identification of clusters using the breadth-first search algorithm [58]. The First step aims to construct a graph in order to represent the data. Indeed, each object in the dataset is represented as a node in the graph. When the similarity measure between two objects is less than a threshold, which is given as an input parameter, an edge is added between them. Therefore, this first step performs the calculation of vertices degree, the calculation of the adjacency lists indices, and finally the assembly of adjacency lists. The second step aims to identify the clusters by traversing the graph created in the previous step using the breadth-first search algorithm. According to the experiments, the G-DBSCAN algorithm is 100× faster than the serial implementation of the DBSCAN algorithm.

Like the methods described above, Mr. Scan [131] is a parallel implementation of the DBSCAN algorithm, which combines the MRNet tree-based distribution network [106] with GPGPU-equipped nodes. The authors of Mr. Scan first of all proposed to optimize DBSCAN with respect to three issues, namely: load imbalance, distributed merge, and data distribution. In order to avoid the load imbalance issue, Mr. Scan finds the densest regions and infers their membership in a cluster without evaluating the points inside these regions. Additionally, the distributed merge issue was addressed by using a number of representative points per cluster to perform the merge

without the need of the whole content of each cluster. And finally, the data distribution issue was addressed by using a heuristic that spatially decomposes the data into roughly equal partitions. Mr. Scan was implemented in such a way that processes are organized into a multi-level tree with arbitrary topology, in which the DBSCAN calculations are done on the GPGPU leaf nodes and the results are combined with non-leaf nodes. The proposed algorithm consists of four phases: partition, cluster, merge, and sweep. The first phase divides the input file in order to create one partition per clustering process. This phase is parallelized using MRNet. The second phase runs in parallel on each leaf node, executing a highly multi-threaded implementation of DBSCAN on a GPGPU, and selecting the representative points. The representative points are sent to the intermediate processes to start the third phase where the clusters are merged to generate the final output clustering. Then, the last phase writes the finalized clusters out to the file system. The experiments have demonstrated that Mr. Scan can perform a clustering of 6.5 billion points on 8,192 GPU nodes in only 17.3 minutes.

Grid-based clustering

Compared to the other techniques of clustering, there are fewer implementations of subspace clustering algorithms on GPU, like GPUMAFIA [1].

GPUMAFIA is a parallel version of the MAFIA subspace clustering algorithm [97] using GPU and CUDA. MAFIA, which stands for Merging of Adaptive Finite IntervAls, applies an adaptive grid method. Indeed, it starts with dividing each dimension into bins, then counting points for each bin and building windows. In this first phase, each window is considered as a candidate dense unit of dimensionality 1 (1-CDU). If the CDUs are dense enough, they become dense units (DUs). The second phase performs the CDU generation, CDU deduplication, finding dense CDUs or point counting, and checks for unjoined DUs. These tasks are computationally demanding. In the last phase, the connected groups of DUs of the same dimensionality are merged into clusters. Before implementing the GPU version of MAFIA, the authors proposed a number of algorithmic changes in order to reduce the computational complexity of this algorithm. This algorithm contains many loops that make its parallelization challenging. For the first phase, the authors parallelized both the dimension and point loops using the shared-memory atomics. For

the second phase, the point counting task consists of a doubly nested loop, the outer over the CDUs and the inner over bit array words. On the GPU version of MAFIA, the loops are mapped into different dimensions of the CUDA thread-block grid. Each thread adds up points from several words of bit arrays in order to count bits in each word, and global-memory atomics in order to compute the final point count. The bit array consists also of a doubly nested loop, the outer over windows and the inner over words or points. As for the point counting, the parallelization was ensured using the global-memory atomics in order to set individual bits. The experiments show that the GPU version of MAFIA provides a further 1-2 orders of magnitude speed-up over a single CPU core or about an order of magnitude over a typical multi-core CPU on typical datasets.

Graph-based clustering

CUDA-MCL [25] is a parallel variant of the Markov clustering algorithm (MCL) [126] applied to the protein-protein interaction networks. The MCL algorithm discovers the cluster structure in graphs by adopting the concept of random walks. The proposed parallel implementation is based on GPU programming via CUDA. The MCL algorithm is based on two algebraic operations on the Markov matrix, namely, the expansion and the inflation. Therefore, the performance of this algorithm depends on the size of the Markov matrix. The main contributions of the authors include the enhancement of the original MCL by implementing parallel tasks for the expansion and inflation operations. In addition, they optimized the data storage using sparse matrix data structures. The proposed algorithm involves three parallel threads CUDA kernels. A kernel to compute the parallel MCL expansion processes, a second kernel to compute the parallel MCL inflation processes, and the last kernel to compute the parallel local and global chaos. Compared to the original MCL algorithm, which is based on CPU, the proposed approach is faster on large datasets.

The Calculation-On-Demand CAST with GPU (COD-CAST-GPU) algorithm [83] is a parallel design of the Clustering Affinity Search Technique (CAST) [20]. As its name suggests, it is based on the GPU platform and the individual memory of the graphics card. The CAST algorithm requires as input the similarity matrix and the affinity threshold. Therefore, the size of such matrix is a critical point which causes the storage problems. The idea behind the proposed

algorithm is to calculate the similarity between data nodes only when it is needed in order to avoid the prior calculation of the similarity matrix. In addition, in order to accelerate the CAST algorithm and gain in terms of performance, the authors exploited the capabilities of the GPU. The proposed algorithm starts by selecting a random node as a new cluster. Then, it uses two operations: *ADD* and *REMOVE* to form this new cluster. After each movement, the affinity values are updated in parallel, and this is where the GPU comes in. The technique adopted by the authors showed remarkable improvements in terms of performance compared to the original algorithm.

Model-based clustering

Async-EM [6] is a parallel variant of the well-known Expectation-Maximization (EM) algorithm based on the GPU platform. Their main contributions concern the synchronization of cores and the organization of memory access. Indeed, the authors noticed that several updates of the global cluster representatives cause inefficient use of memory bandwidth and synchronization overhead. Therefore, to avoid these issues, they proposed to update the global cluster representatives only when a certain number of membership changes have occurred. This is achieved through the idea of the asynchronous model updates combined with an efficient technique, called the model of consolidation. The consolidation model is responsible for merging the different sets of cluster representatives when the local model updates are exchanged. And, it exploits the special characteristics of the memory hierarchy of modern GPUs. The experiments on real and synthetic datasets show that the Async-EM. This method outperforms the incremental-EM and the Batch-EM algorithms in terms of the convergence, the modelling error, and the execution time performances. A comparison between the GPU and the CPU performances of the proposed approach has also shown its effectiveness.

Nature-inspired clustering

A parallel implementation of a nature-inspired algorithm for document clustering is introduced in [145]. The original method is called the flocking-based document clustering algorithm [34]. In order to reduce the execution time and the complexity of this algorithm, the authors exploited the

computational power of Beowulf-like clusters equipped with GPUs. In a typical flocking model, the behaviour of an individual is based only on its flock-mates neighbour within a certain range. This behaviour is described by three rules: separation, alignment, and cohesion. In order to adapt this model to the document clustering problem, the document is considered as an individual that participates in the flocking formation. Then, the three rules are combined for similar neighbour documents, while only the separation rule is applied for non-similar neighbour documents. The authors developed a special GPU cluster programming model to implement the proposed method. It consists of a distributed object interface to unify CUDA memory management and explicit message passing routines. In addition to a mechanism to spawn a flexible number of host threads for parallelization that may exceed the number of GPUs in the system. The GPU cluster programming model contains also an interface for advanced users to control thread scheduling in clusters. The experiments show that the GPU-based implementation scales up over one million documents processed simultaneously in a sixteen-node moderate GPU cluster. Moreover, the proposed method can reach up to 50× speed-up compared to its CPU-based implementation.

2.2.3 Clustering algorithms using Field Programmable Gate Arrays

Partitioning-based clustering

SAKMA [67] is an FPGA-based implementation of the k-means algorithm. This architecture accelerates the whole k-means algorithm through certain approaches such as the pipeline, the tile technique, the duplication parallelism, and the hardware adder tree structures. The SAKMA architecture contains a processing system part in the software and a processing logic part in the hardware. The FPGA-based implementation of the k-means algorithm is carried out by the IP Core. The frequent off-chip memory access is a common problem that the authors resolved using the tiled technique. This technique aims to divide the large volume of memory blocks into small tiles that can be buffered on-chip. The experimental results on real-life biological and synthetic datasets show that the proposed method can reach a speed-up at 20.5× with the affordable hardware cost when compared with other state-of-the-art methods.

In the same context, an accelerated k-means algorithm using the FPGA and a binary tree data structure is proposed in [132]. Indeed, in this work, the authors adopted the filtering algorithm

[70] which intends to reduce the computational load using a kd-tree as the main data structure. This algorithm consists of constructing a tree from the set of points, then it traverses the tree iteratively and updates the centers. The main challenge in this proposed work is how to pipeline and to parallelize the kd-tree processing using multiple banks of distributed on-chip memory. In order to efficiently handle the memory resources the on-chip dynamic memory allocation is used, which allows the allocation of the average amount of memory required during runtime. The proposed method needs five × fewer computational FPGA resources compared to the conventional k-means algorithm implemented in parallel for the same throughput constraint.

The work proposed in [7] is also related to the k-means algorithms. Indeed, it is a k-means-based multi-prototype learning system strengthened by an FPGA-based implementation co-processor for the nearest Euclidean distance searching. This technique allows overcoming the high computational cost of the nearest neighbour searching implementations for the k-means clustering algorithm and the one nearest neighbour (1-NN) classification algorithm. The aim of the proposed system is to construct prototypes using the k-means algorithm iteratively until its convergence to stable centroids. Then, these prototypes are used in order to recognize the test samples by applying the 1-NN classifier to search for the nearest Euclidean distance among these prototypes. Finally, the k prototypes with the best training result are selected to represent the final recognition. The experiments on a handwritten digits dataset show that the proposed learning system can reach an accuracy rate of 97.91% with 930 prototypes.

Another contribution to the k-means algorithm is proposed in [64]. It aims to accelerate k-means for the Micro-array datasets, which are known for their large sizes. The proposed approach requires a careful analysis of samples of Micro-array data in order to fix the values of some critical parameters, such as data size, dynamic range, precision, and memory requirement. The k-means algorithm is implemented using three main blocks. The first block is responsible for calculating distances, the second block is responsible for assigning data points and the third block is responsible for recalculating the centers. Respecting this order, each block executes its operations in parallel and as soon as it finishes it activates the following block. This process reiterates until the convergence. The experiments on a sample of a real-life dataset show that the proposed method can reach a speed-up of up to 51.7× compared to a software model and 206.8×

more energy efficient than the CPU implementation.

Density-based clustering

In order to achieve a high-performance clustering, authors of [108] proposed an FPGA-based implementation of the well-known DBSCAN algorithm. DBSCAN is partially parallelized because the authors considered that the step aiming to obtain the extended neighbourhood of points for a cluster consumes the longest portion of the execution time. And since this step has no data dependencies, it is suitable to be performed in parallel. The proposed method is a fully configurable IP core. Thus, it contains many parameters to be adjusted, such as the size and dimensions of the input data, internal precision, pipeline depths, and the level of parallelism. Performance evaluation based on 2D point clustering shows that the execution time of FPGA-based implementation is not impacted by the increase in the number of dimensions. The proposed method can reach mean speed-ups of 31× in real-life datasets and 202× in synthetic datasets when compared to the state-of-the-art methods.

The study introduced in [110] is also related to the DBSCAN algorithm. It is an FPGA implementation of this algorithm that adopts parallelism at task and data levels. Their main contributions include a design of a data reuse pipeline structure. Such a structure helps overcome the problem of the extra memory access caused by the data dependencies in the parallel algorithm. The second contribution is the use of a robust collision check mechanism in order to prevent wrong clustering results in some special conditions. The experiments on synthetic datasets show that proposed parallel architecture can reach up to 86× speed-up compared to a software implementation on the general-purpose processor and up to 2.9× compared to a software implementation on the graphic processor.

Grid-based clustering

Authors of [119] proposed 2D-clustering implementation based on multi-core FPGA for real-time image processing. The proposed clustering implementation operates on zero-suppressed data and it consists of a pipeline of three modules: the hit decoder module, the grid clustering module, and the centroid calculation module. The first module is a preprocessing step that transforms

the incoming data into a recognized form. The second module identifies the clusters using an innovative moving window technique to reduce the FPGA resources required for this process. Finally, the third module is a post-processing step that performs the data reduction process. In other words, the cluster data are replaced with a single set of centroid coordinates. In order to parallelize this implementation, the authors proposed to instantiate multiple grid clustering modules that work independently on data from separate pixel modules. This is achieved through two logic modules, the parallel data distributor module, and the data merger module. The experiments concerned the 2D-clustering single flow implementation and its parallel version up to 16 clustering engines. A comparison with a previous version showed that the proposed parallel implementation uses 64× fewer logic resources.

2.3 Heterogeneous architectures-based clustering algorithms

Recently, there have been more efforts to improve the efficiency of the parallel clustering algorithms by combining different distributed frameworks in order to obtain more efficient calculation systems. This type of system is referred to as heterogeneous architectures. In that respect, there are several works presented such as [28], [51], [30], [31], and [103].

Partitioning-based clustering

Among the possible combinations of these hybrid architectures, combining the CPU and the GPU platforms. The work presented in [28] is a parallel fuzzy minimal (FM) clustering algorithm for large datasets on Intel and Nvidia-based heterogeneous architectures. The FM algorithm includes two major independent procedures. The first procedure concerns the calculation of the r factor parameter in order to measure the isotropy in the dataset. The second one calculates the prototypes and brings them together when the distance condition is met. Thus, the parallelization of the FM algorithm requires the parallelization of these two main steps based on the CPU and GPU implementations. Regarding the Intel-based parallelization, the process starts by computing the convolution as well as the determinant for each row of the input matrix which are parallelized using OpenMP. Then, the partial factor r computations are combined in order to obtain a final r factor. The next step parallelizes the prototypes via vectorization. Similarly, the CUDA-based

parallelization process starts by calculating the fuzzy covariance matrix, then computes the determinant using the LU factorization, and finally reduces the diagonal in the GPU to achieve the determinant result. In the second step, each thread calculates the distance between each row dataset and the prototype on the GPU, and then a reduction is applied to each block to obtain the total amount for each row. The analysis of the FM clustering algorithm for heterogeneous systems based on Intel CPUs and Nvidia GPUs proves its efficiency as it achieves a speed-up of up to 40 × compared to the sequential version.

Another possible combination of the distributed frameworks is to incorporate the FPGA into spark environment using MapReduce. That is the case of the work suggested in [51], which proposes a k-means library on heterogeneous CPU- FPGA distributed platform by leveraging the spark framework. In that respect, the implementation of this popular clustering algorithm includes two main stages, the data extraction and the data processing. The first stage consists of applying map transformations and distributes the RDD partitions between the processing nodes. These partitions are then transferred into the prellocated buffers in the FPGA's off-chip RAM. In the second stage each node transfers the initial position of the k centroids, and then broadcast the RDD partitions into the FPGA, and center obtains the partial sum and count from each cluster. The last step in this stage consists of updating the k centers by gathering the sum and count values obtained from each node using Spark's framework. The suggested implementation of k-means was compared to the k-means function included in the Machine Learning Library (MLlib) provided with a spark. The results demonstrate the performance of the newly developed platform based on heterogeneous CPU-FPGA and spark framework.

In the context of processing clustering algorithms on heterogeneous CPU-GPU architectures with Flink [26], there are some relevant works developed such as [30] and [31]. FlinkCL [30] is an in-memory computing architecture for heterogeneous CPU-GPU clusters based on OpenCL. The use of the OpenCL allows Flink to take advantage of the computing capacity of GPUs and therefore obtain an obvious programming model. In that respect, the suggested architecture consists of four techniques: a heterogeneous distributed abstract model (HDST), a Just-In-Time (JIT) compiling schema, a hierarchical partial reduction (HPR) and a heterogeneous task management strategy. The parallelization of k-means in FlinkCL architecture is based on two

main steps. The first one starts by grouping the data points as a bulk in the SoA layout, then for each data point find the nearest center using the map function. The cluster centers are considered as broadcast variables which are stored in local memory to speed up their reading. The second step brings together the data points belonging to the same cluster, then calculates the new cluster centers at the end of each iteration using the reduce function. The evaluation of the k-means algorithm with FlinkCL using HPR scheme improves its performance by up to 11 × compared to the sequential version. Similarly to FlinkCL, GFlink [31] is also an in-memory computing architecture based on the heterogeneous CPU-GPU that enhances the computing capacity of Flink. The proposed design involves three techniques: a JVM-GPU communication strategy, a GPU cache scheme, and a flexible locality-aware programming model for three-stage pipelining execution. The main goal of this system is to hide the complexity of GPUs and attain high performance and better load-balance. The k-means clustering algorithm was parallelized and evaluated on GFlink architecture. The results indicate that the k-means algorithm is enhanced by about 5 × and therefore the capacity of GFlink exceeds that of the original CPU-based Flink.

Density-based clustering

As regard the density-based clustering algorithms, the authors [103] developed a new end-to-end system called BD-CATS with the aim of performing clustering at trillion particle scale using parallel hybrid implementation. The system proposed involves various scalable steps. The first step consists of reading data using the HDF5 file format and MPI-IO in collective buffering mode. The second step includes the geometric partitioning, which has as objectives, improving the locality of processing as well as eliminating duplicates inter-node connections. The third one aim, to gather particles from neighboring nodes for use in the clustering step. The next step consists of applying clustering using DBSCAN algorithm with three major stages: Kd-tree construction, Local computation and merging and computing cluster IDs. The last step intended to store results using parallel sorting. All the necessary steps of BD-CATS system are based on optimization techniques in order to ensure scalability and efficient use of memory. These steps were parallelized using both MPI and OpenMP implementations. The experiments conducted on massive particle datasets, concerning Cosmology and Plasma Physics applications, improve the

accuracy, high scalability and efficiency of the proposed system. Consequently, BD-CATS system is an end-to-end scalable solution.

2.4 Comparison of the Parallel Clustering Algorithmss

The research carried out in the context of this survey has highlighted several noteworthy observations. The first point concerns the fact that the majority of the parallel clustering algorithms, which are proposed recently, focused on some well-known algorithms, such as the k-means [59], DBSCAN [45], and OPTICS [9] algorithms. The nature of these algorithms makes them suitable for parallelism in different designs and based on various platforms to handle Big Data.

Comparison baselines

The clustering results of these algorithms, which are used in our tabular comparisons, were extracted from the work of their original authors. Each algorithm is firstly classified according to the Big Data platform used to attain the parallelism of the original algorithm. Then, we present the initial parameters, the type of data processed, and the criteria of clustering validation. These criteria include the quality of the obtained results, the running time consumed, and the speed-up achieved. The quality of the results depends on the data type processed and therefore the measures used differ in some cases. Finally, we present criteria related to the ability of the proposed algorithm to handle Big Data through the volume, the velocity, and the variety of the processed data.

Parallel k-means algorithms

Although k-means is known to be popular and easy to implement, it suffers from some weaknesses such as the determination of the suitable number of clusters k, in addition to the scalability issues when treating sparse values. k-means is an iterative algorithm that involves three main steps: initialization, clusters assignment, and centroids update. There are several designs that may be put forward to perform the clusters assignment and centroids update steps in parallel. Table 2.1 provides a general idea of the performance of the k-means algorithm when implemented using different Big Data platforms. In this table, it can be observed that the MapReduce-based

implementation of the k-means algorithm can process up 4 billion data points in only a few minutes.

Table 2.1: Comparison of the recent parallel clustering algorithms of k-means

	Platforms	Algorithms	Clustering criteria					Big Data criteria		
			Input parameters	Datasets type	Quality	Running time	Speed-up	Volume	Velocity	Variety
Horizontal scaling	Map-Reduce	[35]	K: number of clusters	Real, synthetic	DBI: 0.0133 for WMC 0.0128 for DMC (N = 4b, K = 100, 16 reducers)	5.38 min for WMC 5.29 min for DMC (N = 4b, K = 100, 16 reducers)	–	Up to 4 billion points	No	Data points
	Spark	[124]	ε: threshold	Real	NMI: 0.8412 (N = 621k, ε = 0.5)	9 h (N = 621k, ε = 0.5)	2.6× (N = 621k, ε = 0.5)	Up to 2.8 million points	No	Face videos
	Peer-to-peer	[13]	K: number of clusters	Real, synthetic	AC: 80.14% (N = 35k, K = 7)	–	–	Up to 1 million points	No	Data points
Vertical scaling	GPU	[36]	K: number of clusters D: input size	Real, synthetic	–	–	88× (N = 500k, K = 128, D = 128)	Up to 500,000 points	No	Data points
	Multi-core CPU	[54]	K: number of clusters chunk_size: size of chunks	Real, synthetic	SSE: 7.42E+12 (N = 11m, K = 10)	12.9 s (N = 11m, K = 10)	8.2× (N = 11m, K = 100, 12 cores, chunk_size= 10000)	Up to 11,620,300 points	No	Data points
	FPGA	[67]	K: number of clusters	Real, synthetic	–	238.86 s (N = 10k, K = 300)	20.5× (N =3000, K = 60)	Up to 20,000 points	No	Data points

AC: Accuracy DBI: Davies-Bouldin index DMC: Distribution-based merge clustering N: The size of the tested dataset
NMI: Normalized mutual information SSE: Sum of Squares Error measure WMC: Weight-based merge clustering

Parallel DBSCAN algorithms

DBSCAN is as important as k-means. It aims to discover the clusters and the noise in a spatial database requiring two parameters: Eps and MinPts. The Eps parameter denotes the maximum radius of the neighbourhood from a point, while the MinPts parameter denotes the minimum number of points required to form a dense region. Once these input parameters are set, DBSCAN performs clustering in a two-step approach. First, it starts by selecting an arbitrary point from the dataset that fulfils the core point condition as a seed. Then, it groups all points that are density-reachable from the selected seed including itself, thus obtaining a cluster.

DBSCAN offers several advantages, including finding clusters of arbitrary shapes and handling automatically outliers. And unlike k-means, it does not need to set the number of clusters. However, the estimation of its input parameters is a complex and critical task. This algorithm suffers also from the scalability issues like the rest of the traditional clustering algorithms. The two steps involved in this algorithm can be easily performed in parallel on chunks of the dataset. Table 2.2 presents the performance of the DBSCAN algorithm when implemented using different Big Data platforms. The experiments of the parallel implementations of DBSCAN demonstrated that they reached an interesting speed-up in comparison with the original version algorithm that attains 11× (the MapReduce-based implementation of DBSCAN for 1.2 billion records).

Table 2.2: Comparison of the recent parallel clustering algorithms of DBSCAN

	Platforms	Algorithms	Clustering criteria				Big Data criteria		
			Input parameters	Datasets type	Running time	Speed-up	Volume	Velocity	Variety
Horizontal scaling	Map-Reduce	[61]	ε: distance Minpts: minimum cluster size	Real	58 s (64 tasks, N = 1.2b, ε = 0.001, Minpts = 500)	About 11.29× (64 tasks, N = 1.2b, ε = 0.001, Minpts = 500)	Up to 1.2 billion records	No	Spatial data
	Spark	[56]	ε: distance Minpts: minimum cluster size	Synthetic	1493 min (N = 1m, 512 cores, ε = 25, Minpts = 5)	About 137× (N = 1m, 512 cores, ε = 25, Minpts = 5)	Up to 1 million points	No	Data points
	Peer-to-peer	[107]	ε: distance Minpts: minimum cluster size	Synthetic	67.55 s (N = 100k, 33 nodes)	About 17.5× (N = 100k, 33 nodes)	Up to 100,000 points	No	2D data points
Vertical scaling	GPU	[8]	R: proximity radius MinPts: minimum cluster size	Synthetic	82.9 s (N = 700k, MinPts = 4, R = 0.05)	111.6× (N = 700k, MinPts = 4, R = 0.05)	Up to 700,000 points	No	2D data objects
	Multi-core CPU	[44]	ε: minimum distance μ: minimum cluster size c: number of cores	Synthetic	9.4 s (N = 50k, ε = 120, μ = 7, c = 24)	–	Up to 50,000 points	No	2D fuzzy data objects
	FPGA	[108]	ε: distance Minpts: minimum cluster size	Real	211.88 ms (N = 19,504, parallel elements = 300, Minpts = 80, ε = 25)	33.77× (N = 19,504, parallel elements = 300, Minpts = 80, ε = 25)	Up to 25,000 points	No	Data points

N: The size of the tested dataset

Observations

Most of the parallel clustering algorithms proposed in the literature do not handle real-time data and focus on a single type of data, which limits their adequacy to process Big Data. Indeed, most of the data nowadays are unceasingly produced in real-time, which necessitate continuous processing. Such processing should minimize the storage and computation costs in order to analyse large-scale real-time data. Despite the difficulties encountered with this kind of data, the parallel clustering methods, which are proposed in [63, 119], handle data stream and real-time data efficiently using the GPU and FPGA platforms respectively.

The variety of data is also an important dimension of the Big Data, which is unfortunately difficult to handle in the context of clustering. Indeed, most of the parallel clustering algorithms are designed for numerical data. A few others are specialized in other types of data, such as text data, multimedia data. However, Big Data also takes into account heterogeneous structured, semi-structured, and even unstructured data as it represents the largest proportion. Indeed, clustering unstructured data is a challenging task due to the absence of a recognizable representation. To the best of our knowledge, there is a lack of parallel algorithms for clustering multi-view, heterogeneous, or multi-modal Big Data.

It should also be noted that most of the reviewed algorithms require fixing some initial parameters, which involves a complex beforehand study to decide the appropriate values. Such studies are often calling on information about the data distribution, in addition to considerable efforts and time, which is not always feasible. However, there are very few free-parameter algorithms, which are easy to use, particularly in the context of Big Data, where excessive human interactions should be prevented or minimized when processing data. Therefore, there is a persistent need for parallel clustering algorithms that can meet all the observations mentioned above.

2.5 Conclusion

This chapter covered an in-depth review of the latest parallel clustering algorithms sorted according to the Big Data platforms used. There are two fundamental categories of platforms,

which can handle large-scale data processing. In the first category, we addressed the clustering algorithms based on MapReduce, Spark, and Peer-to-Peer networks. These platforms form part of the horizontal scaling platforms. While in the second category, which is known as the vertical scaling platforms, we focus on the clustering algorithms conceived with Multi-core processors, GPU, and FPGA. All the reviewed algorithms were analysed according to the strategies adopted to ensure the parallelism. This work also includes a detailed comparison of the discussed clustering algorithms based on some common criteria of validation clustering results in Big Data context.

After this thorough study, it is observed that most of the reviewed approaches concern some well-known clustering algorithms, such as k-Means, DBSCAN, and OPTICS. This choice lies in the fact that these algorithms are widely studied and suitable for parallelism in different ways. Also, it was noted that some Big Data platforms are becoming less commonly used in clustering, such as the Peer-to-Peer networks. This is due to the rapid advancements in the field of parallel and distributed computing. These efforts gave rise to new programming models and more powerful hardware that exceed the limitations of the old platforms.

CHAPTER 3

NEW HEURISTIC FOR CLUSTERING CATEGORICAL DATA

This chapter is arranged as follows: Section 3.1 provides an Introduction to the contribution. Section 3.2 discusses some related works. Section 3.3 describes the Correlation clustering and the Relational Analysis formalism. Section 3.4 depicts the main steps of the original Transitive heuristic. Section 3.5 gives a detailed explanation of the proposed heuristic. Section 3.6 involves the experimental results. And finally, the conclusion of the chapter is reported in section 3.7.

3.1 Contribution2: An enhanced version of the Transitive heuristic for clustering categorical data

In spite of the significant efforts that have been made to address the data clustering problem, it still presents some challenging issues, including the difficulty of determining the distance measures for some complex datasets, finding the suitable number of clusters, and choosing initial clusters that require a thorough study to get the most suitable ones, and the difficulty to process some data sets due to the complication of their structure. Furthermore, most of the clustering algorithms developed recently manipulate only numerical attributes, while the algorithms focusing on categorical attributes need further consideration. Indeed, numerical attributes are distinguished by their simple representation, since they have a natural order.

Additionally, the measurement functions for this kind of data are easily defined between data points. Unlike numerical attributes, categorical attributes are characterized by a complicated structure that lacks fundamental geometric properties. The chapter proposes to enhance the $Transitive$ heuristic[117] which aims to process large categorical datasets, by exploiting the advantages offered by the Relational Analysis approach [3]. Indeed, this heuristic automatically detects the maximal number of clusters and the similarity threshold. The distinguishing qualities of the $Transitive$ heuristic include the fact that it is a free-parameter method that performs clustering without specifying the number of clusters. In addition, it adopts a specific cluster structure in order to reduce computational time. Another advantage of this method is its capacity to decrease the cost of calculations and therefore, restrict memory consumption. The proposed heuristic concerns some weaknesses of the original $Transitive$ heuristic, namely, the calculation of the representative and how each data object is processed. This chapter includes our second contribution entitled, E-Transitive: An enhanced version of the Transitive heuristic for clustering categorical data, published at an international conference [116].

3.2 Related Works

This section presents a brief overview of the related work, including some correlation-based clustering algorithms [79][93] developed recently, in addition to the only correlation-based clustering algorithm found in the literature that addresses the issue of categorical data [27]. Levinkov et al.[79]presented a comparative study of four local search clustering algorithms based on the correlation. The first algorithm [73], which is called Greedy Additive Edge Contraction (GAEC), operates with asymmetric treatment to contract the edges considering a maximum positive cost. Then, it computes the sum of costs that must be assigned to the contracted edges. The second algorithm is the Greedy Fixation algorithm (GF)[18]. It proceeds by fixing the edges related to the order of their absolute cost using symmetric treatment of joins and cuts. The time complexity of GF and GAEC algorithms are equal and represent a polynomial complexity that can be considered in practice to be linearithmic. The third algorithm is called Cut, Glue a Cut (CGC) [19]and is performed in two principal phases. The Cut phase consists of the bipartitionning of the graph

recursively, while the Glue and Cut phase proceeds by solving a max-cut problem considering each pair of neighboring clusters. The last studied algorithm in the comparative study is the Kernighan-Lin Algorithm with Joins (KLj) [72]. The main operation of this algorithm consists of changing the boundary between any pair of neighboring clusters, through transformations at a lower cost of decomposition. Another new correlation-based clustering algorithm is described at [93]for blind camera identification by clustering image residuals. The proposed algorithm is performed in two steps. The first step involves clustering on the basis of correlation. It groups the residuals with a very low probability, taking into account the labels of the weights associated with images. The second step consists of merging the basic clusters progressively using an ad-hoc refinement algorithm.

Most of the previous algorithms do not handle categorical data. Indeed, to the best of our knowledge, the only correlation-based clustering algorithm dealing with categorical data is the one proposed by Carbonera and Abdel in [27], called CBK-Modes. As its name suggests, this algorithm combines correlation-based attribute weighting and the basic K-modes algorithm. The proposed algorithm expands the basic K-modes by exploiting the correlation-based attribute weighting approach to calculate the correlation relevance index (CRI). It starts by determining the number of clusters and choosing randomly the initial cluster centers. The next step is to calculate the weight of each attribute and the global weight using the CRI, which is achieved through three steps. Firstly, calculate the frequencies between every two attributes of each categorical value in the determined partition. And then, compute the co-occurrences of each pair of attributes in the appropriate partition. The final step implies the calculation of the CRI for each attribute. After the calculation of the weights, the following stage is to form the different clusters by assigning each object to the closest cluster based on the dissimilarity between this object and the appropriate cluster mode. Afterward, update the modes of the different clusters and then recalculate the weight of each attribute using the CRI. These two main phases are reiterated until there is no change between the old cluster modes and the new ones. The CBK-Modes were compared with five state-of-the-art algorithms considering five real-life datasets. The results show that the proposed algorithm is efficient for clustering categorical data.

3.3 Correlation clustering vs. Relational Analysis formalism

This section briefly provides correlation clustering [17] and the Relational Analysis formalism [3] since the $Transitive$ heuristic belongs to this kind of clustering technique. Both correlation clustering and relational analysis represent an intuitive way of clustering. They perform clustering based on the relationships between objects and without fixing the number of clusters in advance. Let $G = (E, V)$ be a weighted graph, where V is the set of vertices and E is the set of edges, and in which edges are labeled either positive to indicate they are similar or negative to indicate that they are different. In this case, correlation clustering consists of partitioning vertices into clusters that agree as much as possible with the edge labels. This is achieved through the maximization of the number of agreements or the minimization of the number of disagreements. Four basic variants exist, namely, weighted max agreements, weighted min disagreements, unweighted max agreements, and unweighted min disagreements. They are defined respectively as follows:

$$\max_X \sum_{i,j} w_{ij} X_{ij} E_{ij} + \sum_{i,j} w_{ij} (1 - X_{ij})(1 - E_{ij}) \tag{3.1}$$

$$\max_X \sum_{i,j} X_{ij} E_{ij} + \sum_{i,j} (1 - X_{ij})(1 - E_{ij}) \tag{3.2}$$

$$\min_X \sum_{i,j} w_{ij} X_{ij} (1 - E_{ij}) + \sum_{i,j} w_{ij} (1 - X_{ij}) E_{ij} \tag{3.3}$$

$$\min_X \sum_{i,j} X_{ij} (1 - E_{ij}) + \sum_{i,j} (1 - X_{ij}) E_{ij} \tag{3.4}$$

Where X is a collection of clusters, w_{ij} denotes the weight of the edges, and E_{ij} is the edges between two vertices. The correlation clustering is closely related to the Relational Analysis approach. Indeed, as in correlation clustering, the Relational Analysis approach, when applied to clustering, aims to discover clusters by maximizing Condorcet's criterion [95] under four linear constraints that define an equivalence relation. Indeed, let $D = \{d_1, d_2, ..., d_N\}$ be a data set containing N data objects and described by $A = \{a_1, a_2, ..., a_k\}$ a set of k attributes. The Relational

Analysis formalism is summed up by the following system:

$$\begin{cases} \max_X Condorcet(C,X)\sum_{i=1}^{N}\sum_{j=1}^{N} C_{ij}X_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{N} \bar{C_{ij}}\bar{X_{ij}} \\ \forall (d_i,d_j) \in D^2 \quad X_{ij} \in 0,1 \quad binary \\ \forall (d_i) \in D \quad X_{ii} = 1 \quad reflexivity \\ \forall (d_i,d_j) \in D^2 \quad X_{ij} - X_{ji} = 0 \quad symmetry \\ \forall (d_i,d_j,d_k) \in D^3 \quad X_{ij} + X_{jk} - X_{ki} \leq 1 \quad transitivity \end{cases} \tag{3.5}$$

Where C is the collective relational matrix, which is calculated based on the individual relational matrices. The individual relational matrix for an attribute a_l is defined as follows:

$$C_{ij}^{l} = \begin{cases} 1 \quad if \quad a_l(d_i) = a_l(a_j) \\ 0 \quad otherwise \end{cases} \tag{3.6}$$

Where $a_l(d_i)$ denotes the value of the attribute a_l for the data object i. The general term of the collective relational matrix C_{ij} with $(d_i,d_j) \in D^2$ is defined as follows:

$$C_{ij} = \sum_{l=1}^{k} C_{ij}^{l} \tag{3.7}$$

And X is a collection of clusters, which represents the partition to optimize. $\bar{C}$ and $\bar{X}$ denote the complementary of the collective relational matrix and the resulting partition respectively. Generally, the correlation clustering and the Relational Analysis formalism are both considered as an NP-complete problem. For this reason, heuristics and approximation algorithms are the most appropriate solutions for this kind of problems.

3.4 The original Transitive heuristic

The *Transitive* heuristic is a new heuristic for clustering large datasets with categorical attributes, by exploiting the advantages of the Relational Analysis Approach[117]. Thus, the distinguishing qualities of this heuristic include the fact that it is a free-parameter method that

performs clustering without specifying the number of clusters. In addition, it adopts a specific cluster structure in order to reduce the computational time. Another advantage of this method is its capacity to decrease the cost of calculations and therefore, restricting memory consumption. The $Transitive$ heuristic can resolve the clustering problems taking the following form:

$$\begin{cases} \max_X Coef(d_i,d_j)X_{ij} \\ \\ \forall (d_i,d_j) \in D^2 \quad X_{ij} \in 0,1 \quad binary \\ \\ \forall (d_i) \in D \quad X_{ii} = 1 \quad reflexivity \\ \\ \forall (d_i,d_j) \in D^2 \quad X_{ij} - X_{ji} = 0 \quad symmetry \\ \\ \forall (d_i,d_j,d_k) \in D^3 \quad X_{ij} + X_{jk} - X_{ki} \leq 1 \quad transitivity \end{cases} \tag{3.8}$$

The coefficient $Coef(d_i,d_j)$ depends on the contingency criterion used to measure the similarity between data objects. For instance, when it comes to Condorcet's criterion, the problem is formalized according to the Relational Analysis approach (equation 3.5). In the case of correlational clustering, Condorcet's criterion must be replaced by the appropriate criterion which takes into consideration the weights of the graph corresponding to the problem.

The $Transitive$ heuristic proceeds in five main steps: the Initialization step, the Construction step, the Intersection step, the Evaluation step, and the Final step. These steps are detailed in the following description.

- **The Initialization step**: this step consists of choosing randomly an untreated data object from the data set, which will be considered as the representative of the cluster being formed.
- **The Construction step**: in this step, the method assigns similar data objects to the chosen representative using Condorcet's criterion. Each data object can belong to several clusters.
- **The Intersection step**: this step aims to determine the shared data objects among the clusters by computing the intersections between the current cluster and all existing clusters.

- **The Evaluation step**: in this step, the shared data objects are treated in order to decide the most appropriate cluster for each of them. The decision is made based on the contribution of each shared data object to its clusters. Indeed, the shared data object is removed from all clusters and kept only in the cluster with a high value of the contribution.

- **The final step**: when all the data objects are clustered, the process terminates. Otherwise, the process reiterates from the Initialization step.

The $Transitive$ steps are clearly illustrated in Figure 3.1.

3.5 The Enhanced version of the Transitive heuristic

The E-Transitive heuristic is an enhanced version of the $Transitive$ heuristic for clustering large data sets with categorical attributes. The proposed improvements brought to the original heuristic will be explained in what follows.

Let $D = \{d_1, d_2, ..., d_N\}$ be a data set containing N data objects and described by $A = \{a_1, a_2, ..., a_k\}$ a set of k attributes. As with the $Transitive$ heuristic, the E-Transitive heuristic aims to cluster the data objects by optimizing any problem formalized in the form of the equation3.8.

In general, the provided improvements affect the different phases of the $Transitive$ heuristic above-mentioned. Indeed, in the proposed heuristic, the selection of the representative elements is different. The representative is determined randomly only in the first iteration. Thus, from the second iteration, the representative element is determined in the foregoing iteration. In that respect, the first element not clustered in the foregoing iteration is considered as the representative element of the cluster of the current iteration. Another possibility is to choose from the uncustered data objects the data object that contributes the least to the representative of the current cluster. Additionally, in order to simplify the process and avoid the fact that one data object may belong to several clusters, the status of each data object is noted either as clustered or as not clustered. Therefore, the Intersection step used in the $Transitive$ heuristic is eliminated, while the Evaluation and the Construction steps are merged in the proposed E-Transitive heuristic. Figure 3.2 summarizes the main steps of the E-Transitive heuristic.

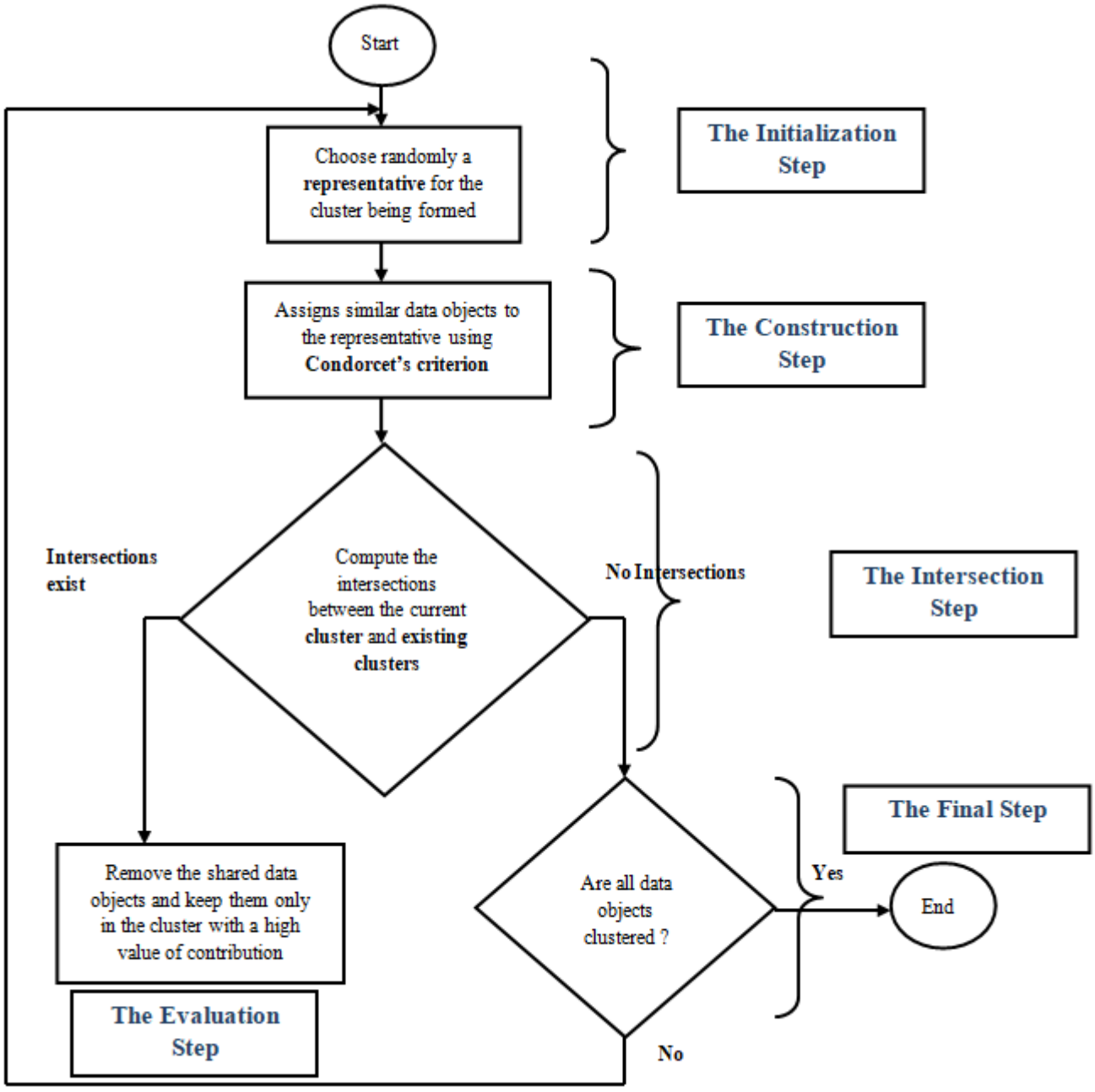

Figure 3.1: The original *Transitive* heuristic flowchart.

In other words, the process starts by browsing the data set sequentially. If the data object being processed is not clustered, the contribution of this data object to the current cluster is calculated using Condorcet's criterion. In the case of a positive value, the data object is added to the current cluster and thus considered clustered. Otherwise, the data object being processed will be saved as the next representative of the next cluster (in the next iteration). In the case

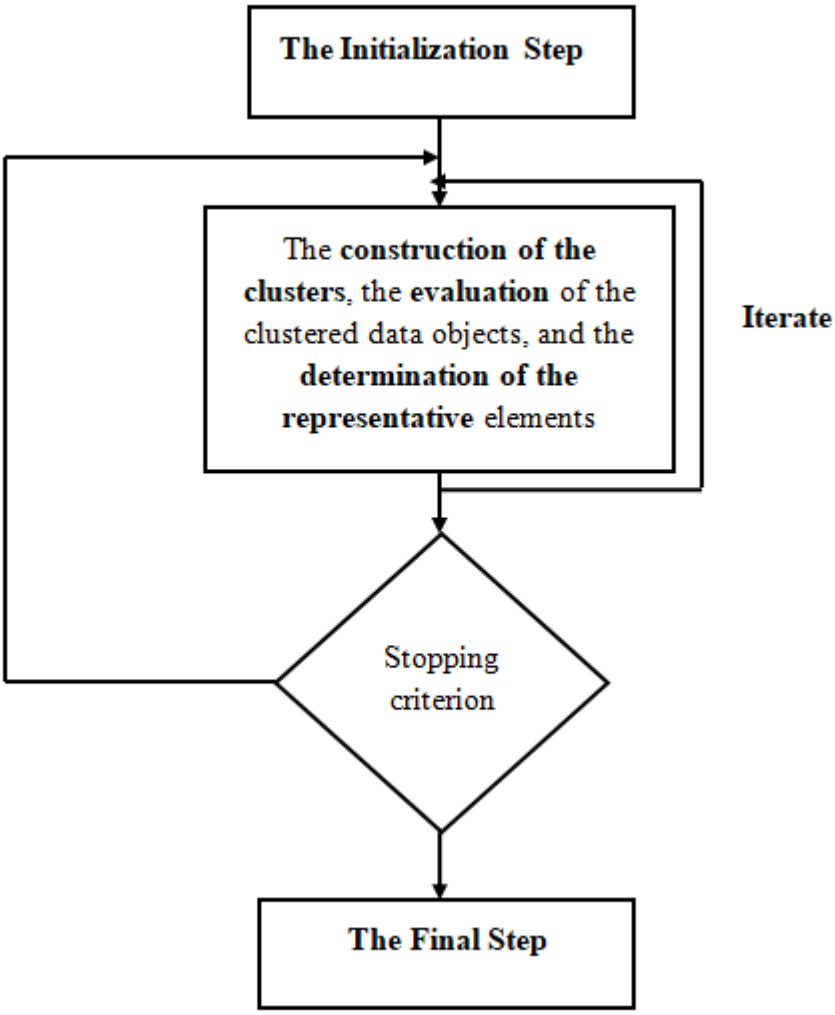

Figure 3.2: The main steps of the E-Transitive heuristic.

where the data object processed is already clustered, its contribution to the current cluster is calculated and compared with its contribution to the cluster to which it already belongs. If this data object contributes better to the current cluster, it will be added as a member to this latter and removed from the old one. Otherwise, no changes will be applied. Another characteristic of this method is that the representatives must be updated after each addition of new members or after a specific number of iterations. The update of the representative is performed using the well-known statistical method called maximum likelihood estimation (LME). In that respect, the complexity of the suggested heuristic is about $\mathcal{O}(kn)$ where k is the number of the obtained

clusters.

The E-Transitive steps are clearly descriped in Figure 3.3.

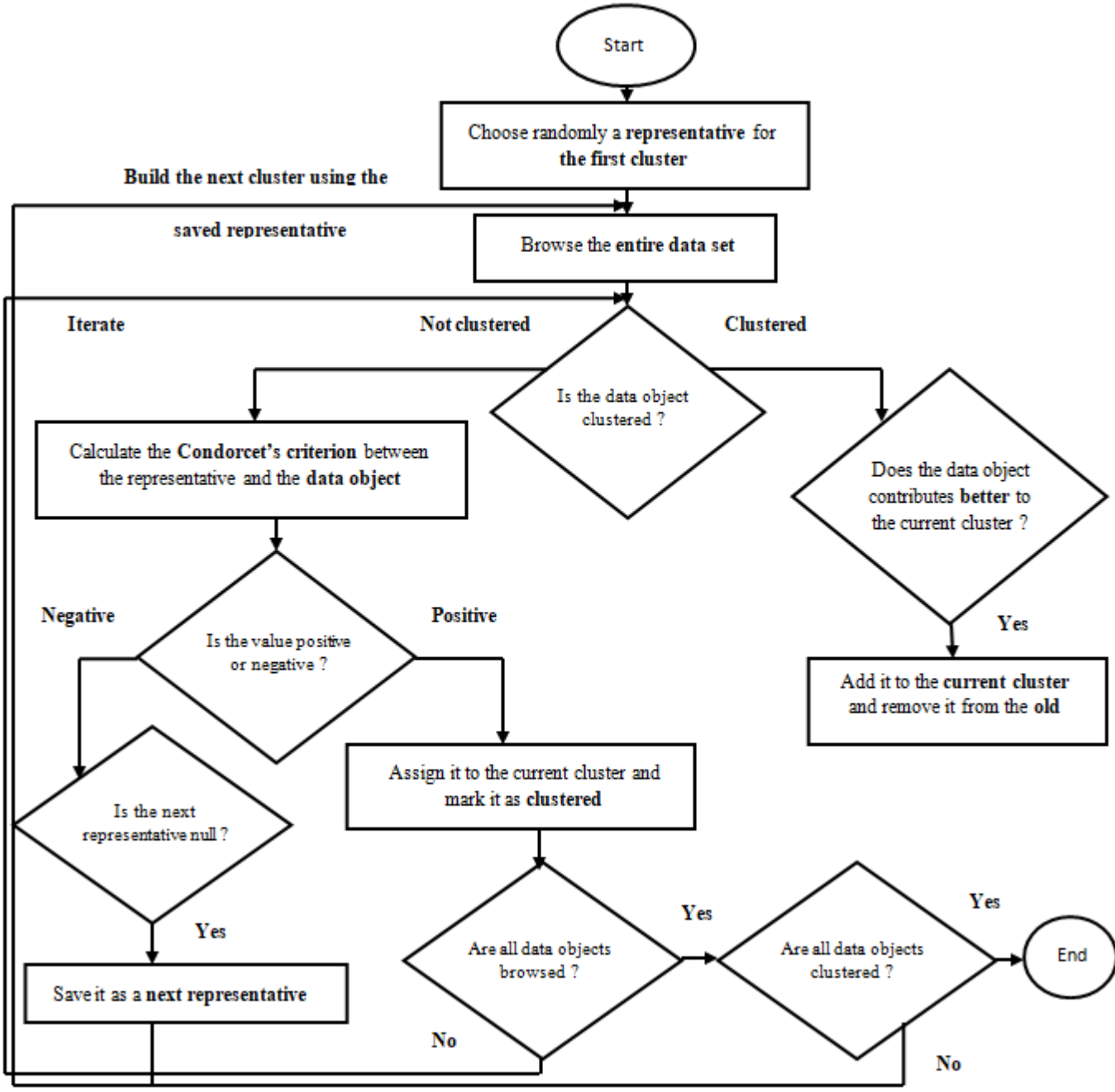

Figure 3.3: The E-Transitive heuristic flowchart.

Let $D = \{d_1, d_2, ..., d_N\}$ be a dataset containing N data objects and described by $A = \{a_1, a_2, ..., a_k\}$ a set of k categorical attributes. The pseudo code of the E-Transitive heuristic will be presented

below.

Algorithm 1 The E-transitive heuristic.

Input:A set of data objects D described by the the set of attributes A.
Output:The data objects in D partitioned in k clusters.
begin

Choose the representative d_r randomly from D
$dnext \leftarrow null$ and $count \leftarrow 0$
while $count < |D|$ **do**
    $dnext \leftarrow null$
    for $i \leftarrow 0$ to $|D|$ **do**
        $d_i \neq d_j$
        if d_i is not clustered **then**
            calculate $Coef(d_i, d_r)$
            if $Coef(d_i, d_r) \geq 0$ **then**
                assign d_i to the current cluster
                update d_r; $count++$
            else if d_{next} is null **then**
                $d_{next} = d_i$
            end if
        else if d_i is clustered in the cluster represented by d'_r **then**
            calculate $Coef(d_i, d_r)$ and $Coef(d_i, d'_r)$
            if $Coef(d_i, d_r) > Coef(d_i, d'_r)$ **then**
                add d_i to the current cluster
                remove d_i from the cluster represented by d'_r
                update d_r
            end if
        end if
    end for
end while
end begin

3.6 Experimentatl results

3.6.1 Data sets description

The E-Transitive heuristic has been tested using three real-life datasets retrieved from UCI Machine Learning Repository, namely, the Zoo, Congressional voting records, and the Mushroom datasets. These latter are the most used in the context of clustering evaluation. Table 3.1 presents a brave description of each dataset tested by the proposed heuristic.

Table 3.1: Description of the real-life data sets used in the experiments.

Data set	Dimension	Classes	Missing values
Zoo	101 × 17	7	No
Congressional voting records	435 × 16	2	Yes
Mushroom	8124 × 22	2	Yes

Table 3.2: Clustering result for the zoo data set.

Cluster ID	Size	Distribution							Accuracy
		C1	**C2**	**C 3**	**C 4**	**C 5**	**C 6**	**C 7**	
1	6	0	0	0	0	0	6	0	**1**
2	8	0	0	4	0	4	0	0	**0.62**
3	4	4	0	0	0	0	0	0	**1**
4	14	0	0	1	13	0	0	0	**0.93**
5	3	0	3	0	0	0	0	0	**1**
6	17	0	17	0	0	0	0	0	**1**
7	37	37	0	0	0	0	0	0	**1**
8	12	0	0	0	0	0	2	10	**0.83**

3.6.2 Results and discussion

In order to test efficiently the suggested heuristic in the different data sets above described, the E-Transitive is executed 100 times since it starts by choosing the first representative randomly. The best result is then selected according to the accuracy of the obtaining results. These results are then compared with the results found by the $Transitive$ heuristic.

Table 3.2 presents the clusters obtained after testing the Zoo data set to the proposed heuristic. The result shows that 5 clusters have reached maximum accuracy value. This means that the distribution found by the E-Transitive heuristic is very close to the clusters that constituted the Zoo data set.

Table 3.4 shows the clusters obtained from the Congressional voting records dataset. Unlike the Transitive heuristic which finds 11 clusters in where the purity values vary from 60% to 100%, the E-Transitive heuristic finds only 10 clusters in where the values of the purity vary from 80% to 100%. As shown in Table 3.4, the most of clusters obtained have reached maximum accuracy. Indeed, the E-Transitive heuristic discovered 13 clusters in the Mushroom dataset of which 10 clusters are 100% accurate and 3 clusters exceed 80%.In that respect, the overall accuracy of the

Table 3.3: Clustering results for the congressional voting records data set.

Cluster ID	Size	Distribution		Accuracy
		Classe 1	**Classe 2**	
1	45	40	5	**0.89**
2	53	50	3	**0.94**
3	5	0	5	**1**
4	26	21	5	**0.81**
5	16	16	0	**1**
6	60	1	59	**0.98**
7	5	1	4	**0.80**
8	112	111	1	**0.99**
9	27	27	0	**1**
10	86	0	86	**1**

Table 3.4: Clustering results for the mushroom data set.

Cluster ID	Size	Distribution		Accuracy
		Classe 1	**Classe 2**	
1	721	40	681	**0.95**
2	774	0	774	**1**
3	296	296	0	**1**
4	1707	0	1707	**1**
5	1296	1296	0	**1**
6	190	190	0	**1**
7	317	290	27	**0.92**
8	423	0	423	**1**
9	1744	1744	0	**1**
10	58	10	48	**0.83**
11	192	0	192	**1**
12	356	0	356	**1**
13	50	50	0	**1**

proposed heuristic reached 98%, while the overall accuracy obtained by applying the Transitive heuristic to the same data set attains97%. Table 3.5 presents the overall accuracy values obtained by testing the Zoo data set, the Congressional voting records data set and the Mushroom data set in order to evaluate the quality of the E-Transitive heuristic and the Transitive heuristic. The comparison between the overall accuracy values shows that the E-Transitive heuristic outperforms the original Transitive heuristic in most cases.

Table 3.5: A comparison of the accuracy of the results obtained using the E-Transitive and the original Transitive heuristic.

Data set	E-Transitive	Transitive
Zoo	93%	91%
Congressional voting records	96%	96%
Mushroom	98%	97%

3.7 Conclusion

This chapter presents the E-Transitive heuristic, an enhanced version of the $Transitive$ heuristic for clustering large categorical data sets. This heuristic is mainly based on the relational analysis model, which allows performing data clustering without specifying the number of clusters. Therefore, the maximal number of clusters and the similarity threshold are automatically detected. The E-Transitive heuristic is then beneficial for clustering large real-world data sets characterized by a complicated structure. How can we better exploit the properties of this heuristic to find effective solutions for the shortcomings of existing clustering methods? And how can we take advantage of its strengths to cluster other types of data?

CHAPTER 4

A PARAMETER-FREE CLUSTERING ALGORITHM USING THE E-TRANSITIVE HEURISTIC ADAPTED TO QUANTITATIVE DATA

This chapter is organized as follows: Section 4.1 presents an introduction to the importance of the E-Transitive heuristic adapted to quantitative data to enhance the k-means algorithm. Section 4.2 introduces some related works. Section 4.3 explains the E-transitive heuristic adapted to quantitative data. Section 4.4 fully describes the different steps of the PFK-means algorithm and its different variants. Section 4.5 provides the experimental results on real-world and synthetic data sets. And finally, section 4.6 covers the conclusion.

4.1 Contribution 3: A Parameter-free clustering algorithm based on k-means

As discussed in the previous chapter, the E-Transitive heuristic aims to cluster categorical data without setting any input parameters. The advantage of this property can be used to solve the initialization issue of the k-means algorithm. Indeed, to benefit from this advantage, an adaptation of the E-Transitive heuristic is necessary to be able to treat quantitative data. In this chapter, a parameter-free clustering algorithm combining the E-transitive heuristic [116] and the traditional k-means algorithm[90][86] is suggested. Indeed, the proposed heuristic does not require the prior initialization of the number of clusters. It progressively generates the initial

cluster centers until the appropriate number of clusters is automatically detected. Hence, the improvements achieved through this heuristic concern primarily two significant weaknesses of the k-means algorithm. The first consists of fixing the number of clusters k, and the second focuses on determining the initial cluster centers. This chapter includes our third contribution entitled, A Parameter-free clustering algorithm based k-means, published in an international journal [115].

4.2 Related Works

Several clustering algorithms have been developed with the goal of ensuring optimal solutions for different clustering problems [21] [105] [66] [142] [143]. k-means is one of the popular partitioned clustering algorithms [90] [86], which aims to cluster a set of data objects into k clusters by minimizing the sum of squared errors over these clusters. Despite its popularity, efficiency, and facility of implementation, the major difficulty encountered with the k-means algorithm is primarily related to its sensitivity to the initialization conditions including the selection of the initial clustering centres, the determination of the number of clusters k, and the possibility to converge on a local optimum [74] [57]. All these aspects influence the quality of clustering. To deal with these issues, researchers devote continuously great efforts to find adequate techniques able to provide suitable initialization parameters, so then ensure a higher clustering quality. Diverse initialization improvements were suggested over the years such as [81] [74] [10][14] [29] [141] [65] [32] [134] [130].

Among these enhancements, the global k-means [81] which is considered as a global search procedure aiming to find an optimal solution for a clustering problem. Indeed, this proposed technique proceeds by adding dynamically one cluster centre at a time using a series of local searches based on fast computed bound on the clustering error. Moreover, it consists of splitting the data space using a k-d tree structure to improve the performance of clustering.

Another initialization strategy represented in cluster center initialization algorithm (CCIA) [74], which intends to perform clustering using two major steps. The first one consists of generating clusters whose number may exceed the number of clusters k. In such a case the second step is

employed by merging the similar clusters using a density-based multi-scale data condensation, and then the merged clusters are treated as the initial cluster centres of the k-means algorithm.

In the same context, the k-means ++ algorithm [10] aims to select the initial cluster centre uniformly at random, then choose the next cluster centres based on a determined probability until the total number of clusters is reached. The next step consists of applying the standard k-means algorithm.

Further enhancement regarding the k-means initialization strategies manifested in the modified global k-means algorithm [14] intents to compute clusters incrementally and determine the k-partition of the data set used based on the previous iterations. Thereby, the algorithm calculates the starting points by minimizing an auxiliary cluster function.

In a similar vein, the authors in [141] develop a new canopy clustering; a pre-processing method for the k-means algorithm, which aims to determine appropriate initial clustering centers and thus attains an optimal number of clusters k. The proposed algorithm covers the pre-processing density canopy method as well as the main k-means processes.

More recently, an entropy-based initialization method [32] for the k-means algorithm was developed to obtain an optimal number of clusters. Indeed, it determines the initial point using the maximization of Shannon's entropy-based objective function, then it aims to detect the best number of clusters based on the optimal cluster detection algorithm for faster convergence.

Another recent work represents the random initialization method [122] merging the bootstrap technique with the data depth concept. Thereby, this method employs k-means with bootstrap replications to find the cluster centres in the original data space. Moreover, it aims to identify a good separation among clusters using depth computation.

4.3 Basic concepts

Suppose a data set X={ x_1, ..., x_n }, containing n data objects in Euclidean space, $x_i \in R^d$, i=1...n. The aim is to partition X into k clusters C_1, C_2, ...,C_k, that is, $\bigcup_{j=1}^{k} C_j = X$ and $C_i \cap C_j = \emptyset$ for $1 \leq i \neq j \leq k$. c_1,c_2,...,c_k are the centres of clusters C_1, C_2, ...,C_k respectively and $c_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$. The difference between c_i, a centre of cluster C_j and a data object x is measured by $dist(x, c_i)$,

where $dist(x,y)$ is the Euclidean distance between two data objects x and y.

The quality of cluster C_j can be measured by the sum of squared error between all data objects x_i in C_j and the cluster centre c_j, defined as:

$$E = \sum_{i=1}^{n} \sum_{j=1}^{k} dist(x_i, c_j)^2, x_i \in C_j \tag{4.1}$$

The average distance of all data objects in the data set$X$ is defined as follows:

$$MeanDist(X) = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} dist(x_i, x_j) \tag{4.2}$$

A new cluster centre $c_{new} \in X$ corresponds to the data object defined by:

$$dist(c_{new}, c_j) = Max(dist(x_i, c_j)) \tag{4.3}$$

where $1 \leq j \leq p$, $p \leq k$, $x_i \in X$, p is the number of the existing cluster centres { C_1, ..., C_p } and c_j corresponds to the jth cluster centre.

4.4 The parameter-free clustering algorithm based on k-means

4.4.1 The E-Transitive heuristic adapted to quantitative data

The E-transitive heuristic [116] is an improved version of the *Transitive* heuristic [117] which aims to cluster categorical data sets using the benefits of the Relational Analysis [3]. In fact, the principal purpose of this heuristic is to perform a clustering without specifying the number of clusters by adopting a specific cluster structure and then reduce the computational time. Thus, with the purpose of processing numerical data, the concept is completely different. More specifically, the property to detect the maximum number automatically is used for clustering numerical data sets. Indeed, the process begins with the calculation of the average of the Euclidean distances of all data objects or a sample of the processed data set using the equation 4.2. The average of the Euclidean distances is considered as a threshold to decide the location of the objects compared to the cluster centers during the construction of the partitions. Then, all data objects are noted as not clustered and each data object added to a cluster is noted clustered.

Accordingly, there are two possibilities. In the case where the data object being processed is not clustered, it will be added to the cluster being formed immediately. Otherwise, in the first step, the distance between the current cluster centre and the data object is calculated and then compared with the distance between the same data object and the centre of the cluster containing this data object. In the case where the data object is most similar to the current cluster, the data object will be removed from the old cluster and added to the overlapping cluster being formed. The cluster centres are updated after each modification. The algorithm 2 presents the instructions of this solution.

4.4.2 The PFK-means algorithm

The proposed heuristic is a parameter-free clustering algorithm, named PFK-means, combining the E-transitive heuristic [116] adapted to quantitative data and the traditional k-means [90][86]. Indeed, PFK-means does not require any initial parameters and generates progressively the cluster centres until the appropriate number of clusters is automatically detected. More specifically, the PFK-means consists of two major stages: the first stage includes the construction of the initial cluster centres and thus discovers the number of clusters k. The second stage consists of applying the traditional k-means algorithm by taking the cluster centres of the first stage as well as the number of clusters detected in the previous stage.

4.4.2.1 The initialization stage

This stage aims to establish the cluster centres without specifying the number of clusters k. In that respect, it starts by calculating the average distance of all data objects using the equation 4.2. Then, it selects the first cluster centre randomly from the data set containing n data objects. The next step consists of calculating the distance between the selected centre and each data object in the data set, using the Euclidean distance. In the case where the distance value is less than the average distance value, the corresponding data object is added to the overlapping cluster, which is being formed. Otherwise, no changes will be applied.

The selection of the other cluster centres follows another strategy. In such a case, the selection is decided during the construction of the forgoing overlapping cluster. The data object the least

Algorithm 2 The E-transitive heuristic adapted to quantitative data

Input:A set of n data objects X
Output:The initial cluster centres Tc_{next}. The number of clusters automatically computed
begin

compute $MeanDist(X)$ or $MeanDist(SampleX)$
select the first cluster centre c_{new} randomly from X
initialize $Next \leftarrow$true
$Tc_{next} \leftarrow$ null
add c_{new} to Tc_{next}
while $Next$ **do**
    $c_{next} \leftarrow$ null
    for $i \leftarrow 0$ to $|X|$ **do**
        calculate $dist(x_i, c_{new})$
        if $dist(x_i, c_{new}) < MeanDist(X)$ **then**
            if x_i is not clustered **then**
                assign x_i to the current cluster
                update the current cluster
            else if x_i is clustered in the cluster whose centre is c_m **then**
                calculate $dist(x_i, c_r) and dist(x_i, c_m)$
                if $dist(x_i, c_r) < dist(x_i, c_m)$ **then**
                    add x_i to the current cluster, remove x_i from the cluster represented by c_m
                    update the current cluster and cluster represented by c_m
                end if
            end if
        else if c_{next} is null **then**
            $c_{next} \leftarrow x_i$
        else if $dist(x_i, c_{new}) > dist(c_{next}, c_{new})$ for all c_{next} in Tc_{next} **then**
            $c_{next} \leftarrow x_i$
        end if
    end for
    $c_{new} \leftarrow c_{next}$
    add c_{new} to Tc_{next}
    if c_{next} is null **then**
        $Next \leftarrow$ false
    end if
end while
end begin

similar to the foregoing cluster centres is defined as the cluster centre of the current overlapping cluster. In other words, the new cluster centre corresponds to the data object defined in equation 4.3. From the second iteration, after the determination of the cluster centre, the construction of the other overlapping clusters is made similarly to the first step. This process continues until all data objects are processed and thus the initial clusters, as well as the overlapping clusters, are obtained. The steps above-mentioned are described in the algorithm 3.

The first stage of the PFK-means algorithm is an independent clustering algorithm that enables the discovery of clusters with appropriate cluster centers. The K-means algorithm is applied to enhance the quality of the clusters obtained.

Algorithm 3 Construction of initial clusters

Input:A set of n data objects X
Output:The initial cluster centres Tc_{next}. The number of clusters automatically computed
begin
compute $MeanDist(X)$ or $MeanDist(SampleX)$
select the first cluster centre c_{new} randomly from X
initialize $Next \leftarrow$true
$Tc_{next} \leftarrow$ null
add c_{new} to Tc_{next}
while $Next$ **do**
    $c_{next} \leftarrow$ null
    for $i \leftarrow 0$ to $|X|$ **do**
        calculate $dist(x_i, c_{new})$
        if $dist(x_i, c_{new}) < MeanDist(X)$ **then**
            assign x_i to the current cluster
        else if c_{next} is null **then**
            $c_{next} \leftarrow x_i$
        else if $dist(x_i, c_{new}) > dist(c_{next}, c_{new})$ for all c_{next} in Tc_{next} **then**
            $c_{next} \leftarrow x_i$
        end if
    end for
    $c_{new} \leftarrow c_{next}$
    add c_{new} to Tc_{next}
    if c_{next} is null **then**
        $Next \leftarrow$ false
    end if
end while
end begin

4.4.2.2 The second stage

The purpose of the initialization stage is to provide the initial cluster centres and detect the number of clusters k automatically. These parameters are the input settings of the traditional k-means executed in this stage. In that respect, the procedure starts by browsing the whole data set and thereafter scrolls through the list of cluster centres provided from the initialization stage and finally assign each data object to the appropriate cluster according to the Euclidean distance. After assigning all data objects to the appropriate clusters, the cluster centres are updated by calculating the mean of the data objects contained in each cluster. The process reiterates until there is no change in the cluster centres values. It should be noted that using the initial cluster centres obtained in the initialization stage as input settings of the traditional k-means allows a rapid convergence and an optimum solution. The pseudo-code of the traditional k-means is described in algorithm 4.

Algorithm 4 The traditional k-means

Input: a set of n data objects X, the list of initial clusters Tc_{next}, the number k automatically computed
Output: the data objects in X partitioned in k clusters
begin

```
repeat
    for i ← 0 to |X| do
        for j ← 0 to k do
            calculate dist(x_i, c_j)
            if dist(x_i, c_j) < MeanDist(X) then
                assign x_i to the current cluster
            end if
        end for
    end for
    update the cluster centres
until Convergence criteria are met
```

end begin

Figure 4.1 resumes the main steps of the PFK-means algorithm.

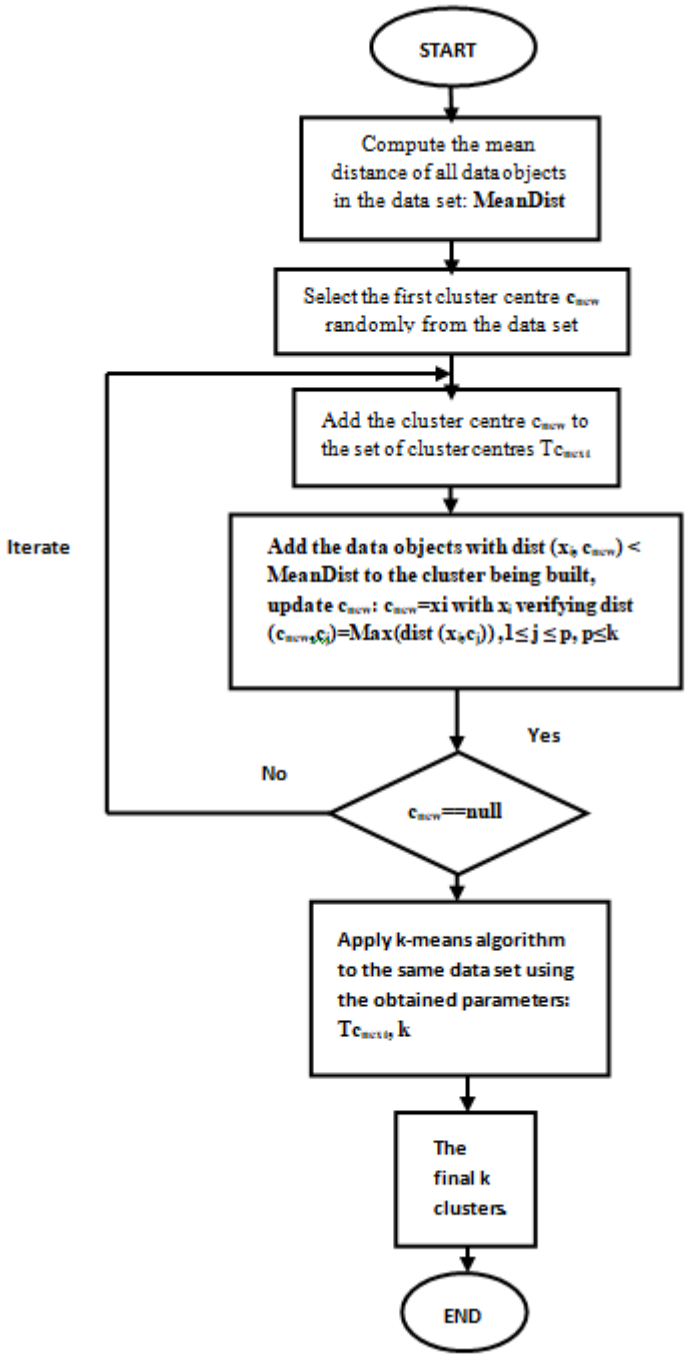

Figure 4.1: The PFK-means algorithm flowchart

4.4.3 Different variants of PFK-means

In order to fully explore the suggested heuristic, several variants of PFK-means have been proposed. These variants mainly focus on the initialization process applied with different approaches: overlapping PFK-means and hard PFK-means.

4.4.3.1 Overlapping PFK-means variant

In the initialization stage, each data object can belong to several clusters and thus the obtained distribution contains overlapping clusters. In that respect, there is one suggested solution with overlapping clusters.

In order to obtain the initial clusters and the number of clusters, the first variant of PFK-means consists of applying the initialization procedure, which is above-explained as a first stage. Thereafter, the second stage starts by browsing the achieved cluster centres and the whole data set and for each data object, calculates the distance between this data object and the current cluster centre based on the Euclidean distance. In the case where the distance value is less than the average distance of all data objects (equation 4.2), the data object being processed is added to the overlapping cluster which is being formed. After scanning the whole cluster centres, each cluster centre value is updated by calculating the mean value of all data objects belonging to its corresponding cluster. This iterative procedure is repeated until no changes occur on the cluster centres values. After the completion of this process, the data set processed is partitioned into k overlapping clusters. The algorithm 5 shows the details of this iterative procedure.

Algorithm 5 The iterative procedure

Input: a set of n data objects X, the k initial cluster centres C, k
Output: the data objects in X partitioned in k clusters
begin

```
repeat
    for r ← 0 to k do
        for i ← 0 to |X| do
            calculate dist(x_i, c_r)
            if dist(x_i, c_r) < MeanDist(X) then
                assign x_i to the current cluster
            end if
        end for
    end for
    update the cluster centres C
until Convergence criteria are met
end begin
```

4.4.3.2 The hard PFK-means version I

This solution consists of applying the initialization process ((algorithm 3) presented in the PFK-means heuristic as a first stage. Then, similarly to the steps explained on the iterative procedure (algorithm 5) assign each data object to the appropriate cluster and in parallel remove the intersections between the constructed overlapping clusters. In other words, when the data object which is being processed is not clustered, it is added immediately to the cluster being formed. Otherwise, when the data object is already clustered, the distance between the current cluster centre and the data object is calculated then compared with the distance between the same data object and the centre of the cluster containing this data object. In the case where the data object is most similar to the current cluster, it will be removed from the old cluster and added to the overlapping cluster being formed. Thus, the cluster centres are updated after each iteration by calculating the mean value of the data objects assigned to each cluster. The algorithm 6 illustrates the steps of this hard iterative procedure.

Algorithm 6 The hard iterative procedure

Input: a set of n data objects X, the k initial cluster centres C, k
Output: the data objects in X partitioned in k clusters
begin

```
repeat
    for r ← 0 to k do
        for i ← 0 to |X| do
            if x_i is not clustered then
                calculate dist(x_i, c_r)
                if dist(x_i, c_r) < MeanDist(X) then
                    assign x_i to the current cluster
                end if
            else if x_i is clustered in the cluster whose centre is c_m then
                calculate dist(x_i, c_r) and dist(x_i, c_m)
                if dist(x_i, c_r) < dist(x_i, c_m) then
                    add x_i to the current cluster, remove x_i from the cluster represented by c_m
                end if
            end if
        end for
    end for
    update the cluster centres C
until Convergence criteria are met
end begin
```

4.4.3.3 The hard PFK-means version II

The process of this variant is similar to that of the first version of the hard PFK-means (algorith 3+ algorithm 6), the only difference is that the last stage of this solution consists of applying the traditional k-means at the end of the process. In that regard, the second version of the hard PFK-means consists of three major stages. The first stage aims to discover the initial clusters by applying the initialization phase as presented in the PFK-means (algorithm 3). Then, the second stage consists of executing the hard iterative procedure as explained in the above variant (algorithm 6). Finally, in the last stage, the traditional k-means is applied by taking the cluster centres and the number of clusters, obtained from the second stage, as input parameters (algorithm 4).

4.4.3.4 The hard PFK-means version III

In a similar vein, this solution starts by applying the initialization stage (algorithm 3). Secondly, it executes the iterative procedure (algorithm 5). At last stage, it runs the traditional k-means algorithm taking as input settings the output parameters of the second stage, which are the initial cluster centres of the obtained overlapping clusters and the number of overlapping clusters automatically computed (algorithm 4).

4.5 Results and Discussion

This section provides the results obtained by implementing the PFK-means heuristic, its different variants, the E-transitive heuristic [116] adapted to quantitative data, and the traditional k-means [90][86] using real-world data sets, retrieved from the UCI Machine Learning Repository [12]. In order to measure the clustering effect, these algorithms are evaluated based on the accuracy and the sum of squared errors described by equation 4.1. The experiments include also the simulation tests which have been performed to evaluate the performance of PFK-means heuristic in terms of running time with distinct synthetic data sets generated using a data mining generator, called weka [55].

Table 4.1: Description of the real-life data sets used in the experiments.

data set	Data size	Attributes	Cluster number
Iris	150	4	3
Wine	178	13	3
Soybean-small	47	35	4
Pima Indian Diabetes	768	8	2
Seeds	210	7	3
Musk	6598	168	2
Letter-Recognition(LR)	20000	16	26

4.5.1 data sets description

Table 4.1 gives a brave description of seven real-world data sets, retrieved from the UCI machine learning [12], used to evaluate the performance of the proposed heuristic, namely, Iris, Wine, Seeds, Pima Indian Diabetes, Soybean-small, Segmentation, Musk, and Letter-Recognition (LR). As shown in table 4.1, each data set is described by a specified number of clusters, many data objects, and each data object is described by a vector of attributes. The simulated data sets are generated by varying the size of the data sets, the number of clusters, and the number of attributes. Indeed, the first experiment consists of generating data sets with different sizes: 1000, 1500, 2000, 2500, 3000, 3500, and 4000. Each of these data sets is described by three clusters and five attributes. In the second experiment, the data set size is fixed at 1000, the number of attributes at 5, and the size of the cluster is varied as follows: 2, 3, 4, 5, 6, 7, 8, 9, and 10. Finally, the last experiment use data sets with a different number of attributes: 5, 10, 15, 20, 25, 30, 35, 40, and fix the data set size and the number of clusters at 1000 and three respectively. Besides, two-dimensional synthetic data sets [47] were used for comparing the suggested heuristic and the iterative k-means minus-plus [65]. These data sets contain 5000 data poins and 15 clusters: S1, S2, S3, and S4.

4.5.2 Clustering evaluation measures

Clustering validation is an important aspect to evaluate the quality of clustering results. Indeed, it depends on some parameters such as the similarity measure, the implementation of the clustering algorithm used, and the capacity to catch some or all of the hidden patterns. In order

to measure the clustering effect of the proposed heuristic, the following parameters are involved: the time required for completing the procedure of clustering, the sum of squared errors (equation 4.1), the accuracy, and the entropy clustering measure.

4.5.3 Results on real-world data sets

In order to evaluate the performance of the developed heuristic and its variants, these heuristics have been programmed using java. The results presented are the best values obtained from 5 runs for each proposed heuristic, except for the second version of the hard PFK-means which produces stable results. Concerning the traditional k-means algorithm, the initial centres were generated randomly.

Table 4.2 provides the sum of squared errors for PFK-means, the E-transitive heuristic [116] adapted to quantitative data and its variants on real-world data sets. Clearly, the second version of the hard PFK-means exceeds the other proposed heuristics in terms of the sum of squared errors for all data sets except the soybean data set. In this case, the PFK-means and the second version of the hard PFK-means give the best results. The performance of the E-transitive heuristic comes back to the fact that this variant makes it possible to detect outliers. Absolutely, since in the iterative procedure applied as the second stage of this heuristic, the data objects which are very far from the cluster centres are not assigned imperatively to these clusters. Therefore, the E-transitive heuristic adapted to quantitative data provides the minimal values of SSE. Additionally, the second version of the hard PFK-means produces stable results. Finally, it should be noted that PFK-means and its variants lead to finding the right number of clusters for all tested data sets as described in table 4.4. Furthermore, regarding the PFK-means heuristic and its hard variants, all inputs are clustered.

In the Figures (Figure 4.2, Figure 4.3, Figure 4.4), a comparison of the clustering results of PFK-means, the E-transitive heuristic adapted to quantitative data, its variants, and the traditional k-means on real-world data sets in term of accuracy is illustrated. Figure 4.2 presents a comparison between the PFK-means and the E-transitive heuristic. Based on this result, it is clear that the accuracy of these heuristics is closed to each other. Moreover, Figure 4.3 describes the accuracy of PFK-means and its variants for the above-mentioned real-world data sets. As

Table 4.2: The sum of squared errors of the clustering results on real-world data sets.

data set	PFK-means	E-transitive	Hard PFK-means1	Hard PFK-means2	Hard PFK-means3
Iris	78.94	**76.46**	82.86	78.94	78.94
Soybean	**205.96**	207.49	220.05	207.49	207.05
Wine	2.63E+06	**2.31E+06**	2.97E+06	2.37E+06	2.37E+06
Pima	5.18E+06	**4.31E+06**	5.68E+06	5.12E+06	5.14E+06
Seeds	**587.31**	**587.31**	630.78	**587.31**	588.43

can be seen, the accuracy of PFK-means and its variants are closed to each other for iris, Pima Indian Diabetes, seeds, and soybean data sets, yet for wine data set the second version of the hard PFK-means and the third version of the hard PFK-means outperform the PFK-means heuristic and the first version of the hard PFK-means. The last Figure (4.4) describes the accuracy of the PFK-means heuristic and the k-means algorithm with UCI data sets [12]. From this figure, it can be shown that the PFK-means outperforms the k-means algorithm for Pima Indian Diabetes and seeds data set. However, for wine and soybean data sets, the k-means algorithm achieves an accuracy superior to the accuracy of PFK-means.

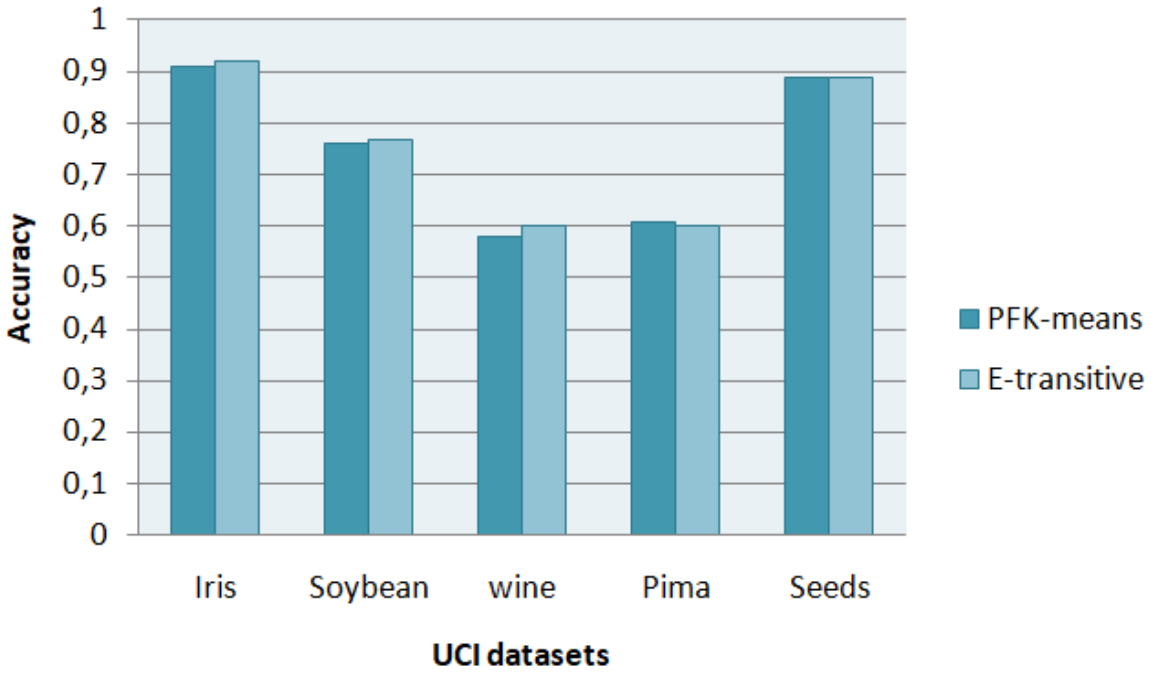

Figure 4.2: The accuracy of PFK-means and E-transitive on real-world data sets.

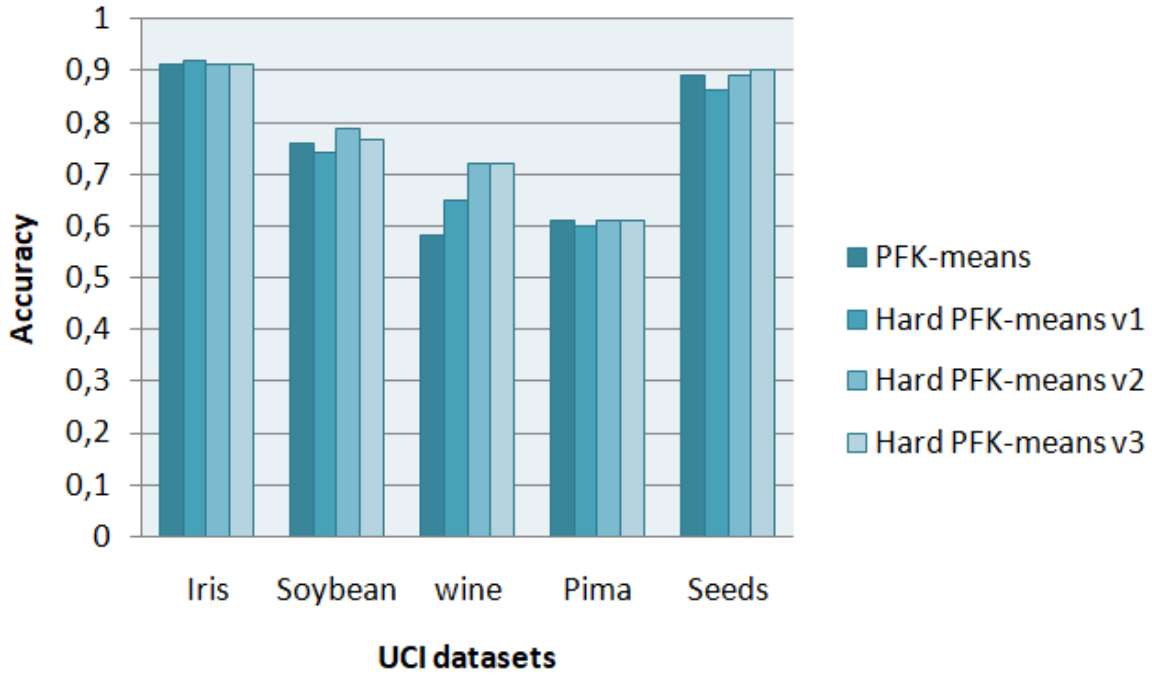

Figure 4.3: The accuracy of PFK-means and its variants on real-world data sets.

Table 4.3 presents a comparison between the PFK-means heuristic and the traditional k-means algorithm [90][86], in terms of a sum of squared errors, accuracy, and entropy measure based on real-world data sets. Regarding the sum of squared errors and the entropy clustering measure, the smaller their values, the better the result. The highest value provided by the entropy clustering measure is one while the lowest one is 0. The PFK-means heuristic exceeds the traditional k-means algorithm in terms of the sum of squared errors for all data sets except the Pima data set. Concerning the accuracy, the PFK-means heuristic gives the best results for Iris, Pima, and Seeds data sets. Furthermore, the values obtained by the entropy clustering measure are the best for the PFK-means heuristic. In addition to that since the suggested heuristic doesn't require the number of clusters as an input parameter, it shows important results compared to the traditional k-means. Thus, the results exposed in Table 4.3 demonstrate the efficiency of the PFK-means heuristic and its ability to find the appropriate number of clusters for all data sets. Table 4.4 presents the number of clusters found after executing the PFK-means as well as the exact number of clusters for the real-world data sets used.

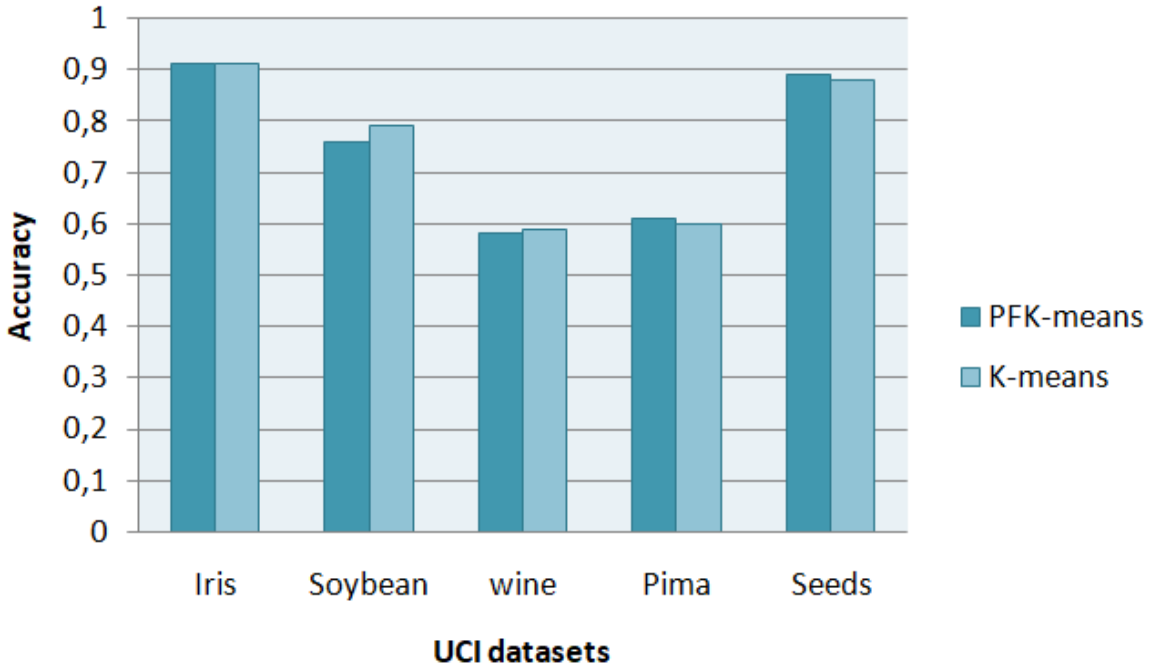

Figure 4.4: The accuracy of PFK-means and the traditional k-means on real-world data sets.

Table 4.3: Comparison of PFK-means and k-means on real-world data sets.

	K-means			PFK-means		
	SSE	**Accuracy**	**Entropy**	**SSE**	**Accuracy**	**Entropy**
Iris	78.94	0.91	0.39	78.94	0.91	0.39
Soybean	208.15	**0.79**	0.56	**205.96**	0.76	**0.55**
Wine	2.66E+06	**0.59**	0.98	**2.63E+06**	0.58	**0.95**
Pima	**5.13E+06**	0.60	0.91	5.18E+06	**0.61**	**0.77**
Seeds	593.50	0.88	0.46	**587.31**	**0.89**	**0.44**

4.5.4 Results on synthetic data sets

For the purpose of testing the performance of the proposed heuristic in terms of processing time, different synthetic data sets were generated based on different clustering criteria namely, the size of the data sets, the number of clusters, and the number of attributes.

The first experiment (Figure 4.5) describes the performance of the proposed heuristic when increasing the size of the data sets, which varies from 1000 to 4000 while setting the number of clusters at 3 and the number of attributes at 5. As shown in Figure 4.5 the running times of the proposed heuristic vary from 188 to 4321 milliseconds which are nearly linear against the size

Table 4.4: The number of clusters obtained after executing PFK-means and its variants.

data set	Exact Cluster number	Cluster number found
Iris	3	3
Wine	3	3
Soybean-small	4	4
Pima Indian Diabetes	2	2
Seeds	3	3
Musk	2	2
Letter-Recognition(LR)	26	26

of the data sets. The next experiment (Figure 4.6) depicts the suggested heuristic behavior by increasing the number of clusters from 2 to 10 with the data set size set to 1000 instances and the number of attributes fixing to 5. It is clear from Figure 4.6 that the clustering time scales linearly from 141 to 217 milliseconds while increasing the number of clusters. Additionally, the proposed heuristic can detect the adequate number of clusters of each generated data set. The last experiment (Figure 4.7) illustrates the processing times in milliseconds while increasing the number of attributes from 5 to 40 with the number of clusters fixing to 3 and the size of the data set setting in 1000. This Figure (4.7) shows clearly that the variation of the running times of the proposed heuristic from 169 to 489 milliseconds while increasing the number of attributes is quite linear.

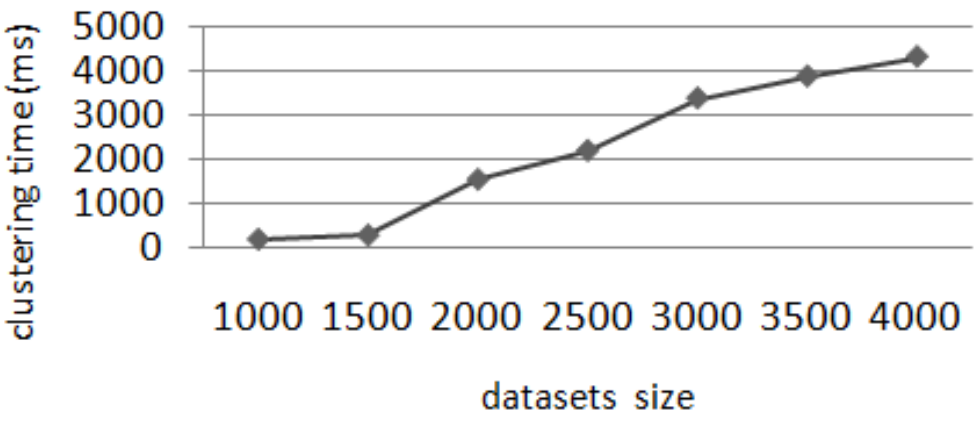

Figure 4.5: Clustering time of synthetic data sets for different sizes.

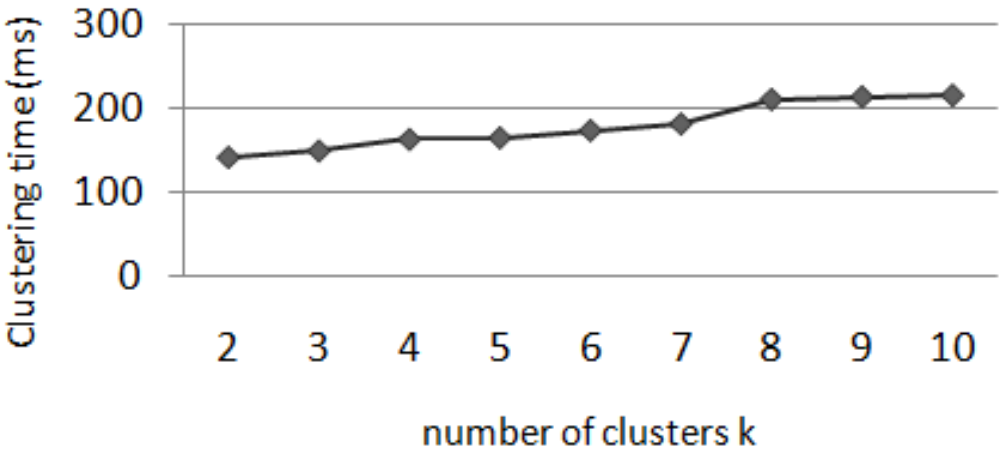

Figure 4.6: Clustering time of synthetic data sets for different clusters.

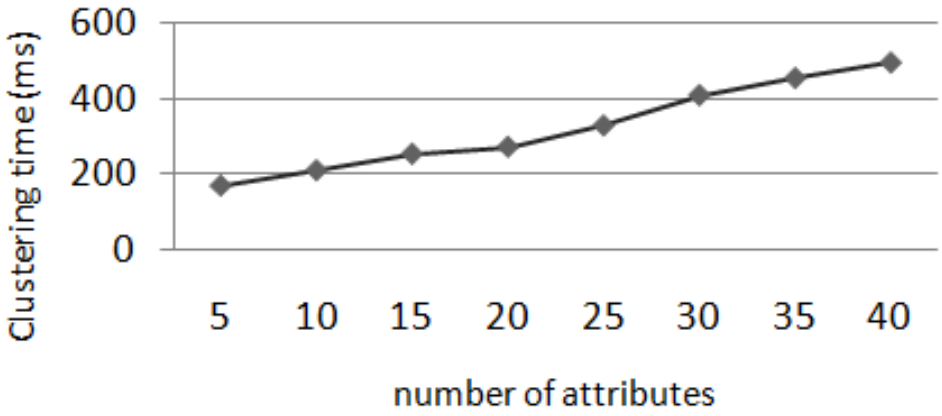

Figure 4.7: Clustering time of synthetic data sets by varying the number of attributes.

4.5.5 Results of PFK-means compared to the iterative k-means minus-plus

The PFK-means heuristic was compared to the iterative k-means minus-plus [65], which is an iterative approach to improve the quality of the k-means algorithm by removing one cluster (minus), dividing another one (plus), and applying re-clustering again, for each iteration. The results of the iterative k-means minus-plus and the traditional k-means were presented as in the original paper describing the iterative k-means [65]. Table 4.5 presents a comparison between the

Table 4.5: Comparison of PFK-means, the iterative k-means-+, and the k-means algorithm on different data sets.

	maximum of partial SSE			SSE		
	KM	IKM-+	PFKmeans	KM	IKM-+	PFKmeans
Iris	6.53E+01	3.98E+01	3.98E+01	9.95E+01	7.89E+01	7.89E+01
Musk	4.61E+09	4.35E+09	4.35E+09	6.09E+09	5.92E+09	5.92E+09
LR	4.17E+04	3.93E+04	3.93E+04	6.20E+05	6.16E+05	6.16E+05
S1	6.59E+12	8.54E+11	8.54E+11	1.85E+13	8.92E+12	8.91E+12
S2	5.78E+12	1.33E+12	1.33E+12	2.01E+13	1.33E+13	1.32E+13
S3	3.41E+12	1.56E+12	2.97E+12	1.94E+13	1.69E+13	2.06E+13
S4	2.56E+12	1.79E+12	2.77E+12	1.70E+13	1.57E+13	1.98E+13

PFK-means heuristic, the iterative k-means minus-plus, and the traditional k-means algorithm, in terms of a sum of squared errors, and maximum of partial SSE for three real-world data sets and four synthetic data sets. The PFK-means heuristic and the iterative k-means minus-plus algorithm outperforms the traditional k-means, except for S3 and S4 data sets when the iterative k-means -+ outperforms the proposed heuristic.

4.5.6 Discussion

From the experiments that have been conducted on real-world data sets, it has been proven that the suggested heuristics can detect the appropriate number of clusters independently of any initial conditions. Accordingly, these heuristics can be successfully used for unsupervised learning. Furthermore, the examination conducted on synthetic data sets demonstrates that the proposed heuristic finds the appropriate number of clusters in reasonable processing time against the variation of the size of the data sets, the number of clusters, and the number of attributes.

4.6 Conclusion

The purpose of this chapter is to present a new clustering algorithm namely a parameter-free clustering algorithm based on k-means. This hybrid solution combines the E-transitive heuristic adapted to quantitative data and the k-means algorithm to deal with the major issue encountered with k-means, which is the determination of the number of clusters and the initial cluster centres. The PFK-means and its variants were explained according to the clustering approaches. Also, this

chapter covers a detailed comparison between the PFK-means heuristic, its different variants, the revisited version of the E-transitive heuristic, the iterative k-means minus-plus, and the k-means algorithm in terms of the sum of squared errors and accuracy

CHAPTER 5

THE PFK-MEANS ALGORITHM: FROM CENTRALIZED TO DISTRIBUTED SYSTEMS

This chapter is organized as follows: Section 5.1 explains the necessity of moving to the parallel version of PFK-means. Section 5.2 introduces some related works. Section 5.3 presents the design of the parallel implementation based on the Spark framework. Section 5.4 describes the experimental results on real-world and synthetic data sets. Section 5.5 provides the conclusion of the chapter.

5.1 Contribution 4: An efficient parallel algorithm for clustering Big Data based on the Spark framework

With the purpose of enhancing the efficiency of the sequential PFK-means algorithm and leveraging distributed systems for processing Big Data. This chapter presents a parallel implementation of the PFK-means algorithm well described in the previous chapter, using open-source Hadoop, the Spark framework with RDDs, and a machine learning-based model. More specifically, the proposed algorithm aims to parallelize the two principal stages of the PFK-means algorithm using Spark RDDs functions. The first one consists of applying successive RDDs functions in order to perform the computation of the Euclidean distances, build the initial clusters, and finally update the different clusters to obtain the cluster centers and, consequently, the number k. The

second stage invokes the use of the parallel K-means provided by Spark mLlib with as input parameters the cluster centers and the number of clusters discovered in the previous phase. This chapter includes our fourth contribution entitled, An efficient parallel algorithm for clustering Big Data based on the Spark framework, being published in an international journal in an international journal (Z. Dafir and S. Slaoui, 2022).

5.2 Related works

Over the years, clustering algorithms have undergone several evolutions and improvements to face the challenges of managing heterogeneous and complex data generated from multiple and varied digital sources. In particular, K-means [90] [86] is one of the most widely used clustering algorithms due to its simplicity and efficiency in processing data. This algorithm has experienced several adaptations and improvements to address initialization issues, exploit distributed systems and handle Big Data [4] [11][23]. In this context, several research studies based on different parallel computing architectures have been carried out to give birth to powerful and scalable algorithms.

Among these algorithms, a parallel adaptive Canopy-K-means algorithm [133] aims to solve the manual selection problem for the distance threshold T2 in the Canopy process. In this regard, it is improved by adaptive parameter estimation using the MapReduce-based Model and the Spark framework. The clustering results on the Stanford Large Network Dataset Collection (SNAP) data set and self-built Dimension demonstrate the efficiency of the proposed solution.

Another improvement in the parallel K-means is introduced in [87], which is an improved K-Means Clustering Algorithm for Big Data Mining based on the density of data points to construct the corresponding clusters. Thus, the algorithm was parallelized on the Hadoop platform using the distributed database to improve its efficiency and decrease the running time. The simulation results prove that the enhanced solution outperforms the classical K-means and the DBSCAN algorithm in terms of clustering accuracy by 10%.

In the same context, a clustering center selection method [88] was developed to solve the issue of the random nature and limited quality of initial cluster center selection. In addition, a

parallel implementation using the Spark computing framework was suggested to perform Big Data clustering and therefore obtain higher execution efficiency, accuracy, and good stability in big data.

More recently, another improved initialization method [85] for the K-means algorithm was implemented to enhance the initial points selection strategy using sparse reconstruction. Moreover, the parallel version of the algorithm was performed based on the MapReduce framework and Hadoop cluster using the real customer data from the JD Mall. The experimental results demonstrate its efficiency in processing large amounts of data.

Another contribution aiming to address the K-means algorithm initialization issue is a parallel clustering algorithm based on grid density [92]. Indeed, the process of the algorithm is based on specific strategies, including, locality sensitive hash function(DP-LSH), the adaptative grouping strategy (AGS), and the MapReduce framework to operate the cluster centers in parallel and therefore increase the performance of the proposed approach.

5.3 Design of the parallel PFK-means using the Spark framework

The PFK-means is a sequential and iterative clustering algorithm, suitable for processing unsupervised machine learning data sets. However, the efficiency of this algorithm will be restricted when dealing with a large amount of data. Among the solutions to improve the performance of this algorithm and make it suitable for processing Big Data is the parallelization of its tasks using the open-source Hadoop, Spark RDD, and Machine Learning-based Model.

In particular, Spark is a horizontally scalable cluster computing framework [139], performing parallel processing for data-intensive applications with working sets. These applications are managed by the driver program, which allows them to execute the user's main function as well as the parallel operations in a cluster.Figure 5.1 highlights the basics of spark architecture.

The implementation of the parallel version of PFK-means consists of the parallelization of its

two main stages. In this way, the first stage consists of the parallelization of three principal tasks, including the calculation of the total average of the Euclidean distances, the construction of the initial clusters, and assigning the data objects to the appropriate clusters as well as updating the discovered cluster centers. The next stage involves the use of the parallel K-means provided by spark mLlib, using as input parameters the discovered cluster centers as well as the number of clusters automatically detected in the first stage. Figure 5.2 illustrates the design of the parallel implementation of the PFK-means. In the following, the steps of the initial stage will be thoroughly described.

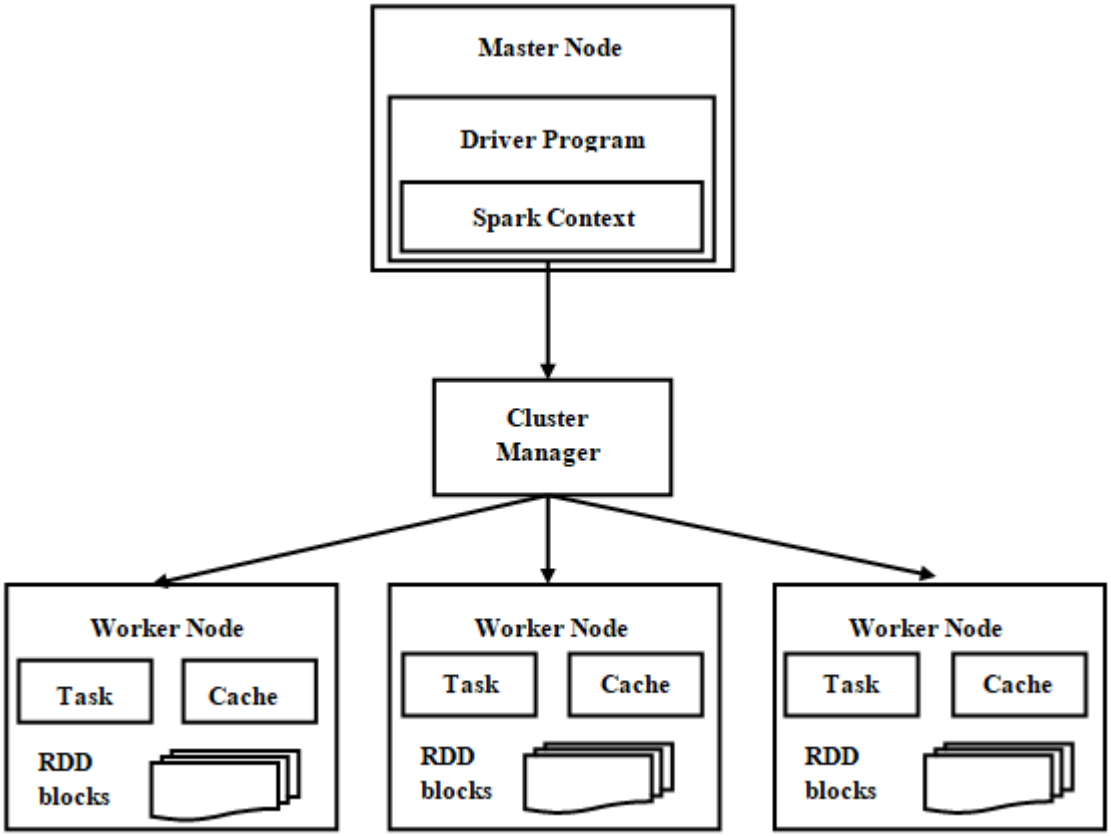

Figure 5.1: Spark Architecture

5.3.0.1 Calculation of the total average of the Euclidean distances

In order to compute the total average of the Euclidean distances, the process starts by reading the data set being processed locally into RDDs partitions since it is the first step. Then these RDDs'

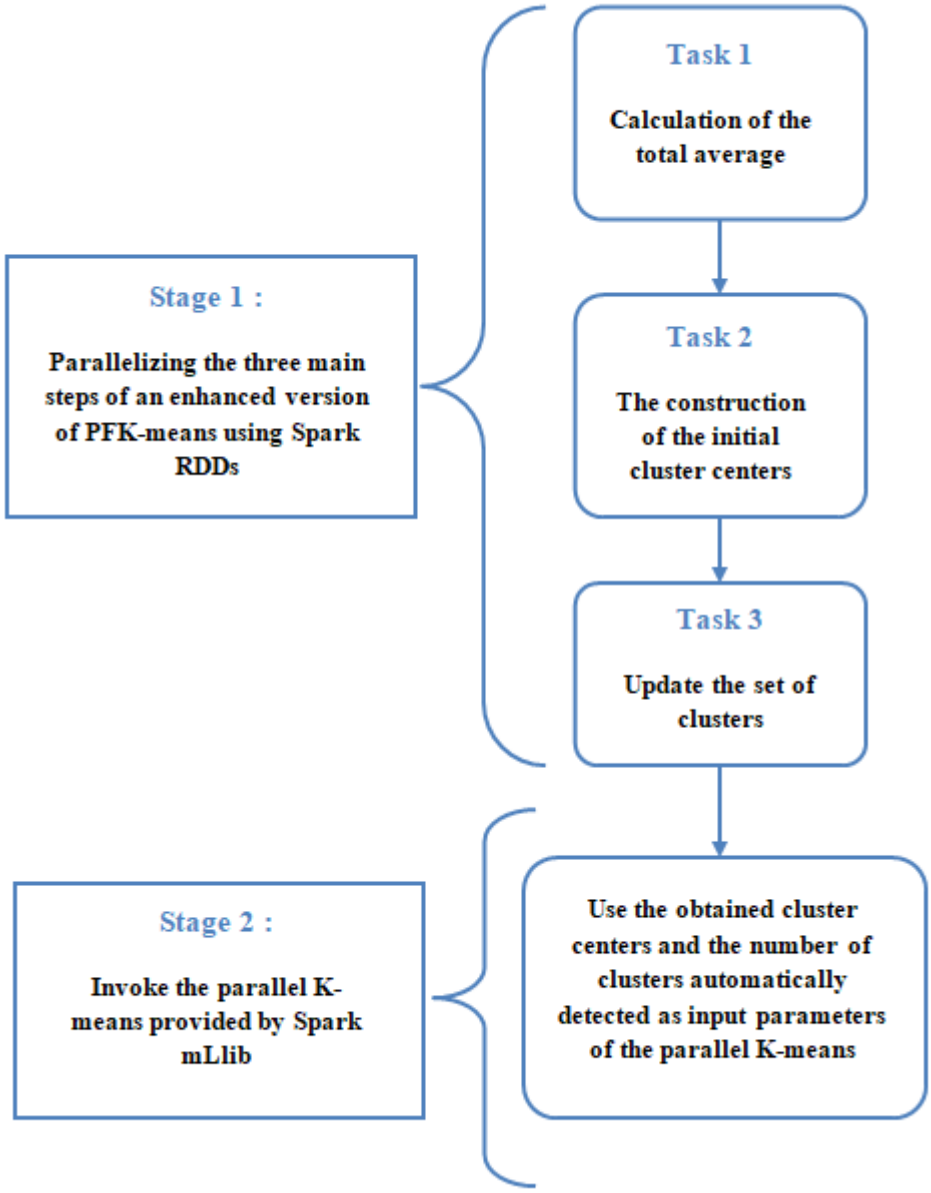

Figure 5.2: The design of the parallel implementation

partitions are transformed using the Map function with the purpose of calculating the Euclidean distance between each two data objects, which is performed in several iterations. Therefore, in each iteration, the Reduce function sums the partial Euclidean distances for each partition Map. Then, after completing all the iterations, another Reduce function is applied to the final result, in order to obtain the total average of the Euclidean distances. Figure 5.3 depicts the process of

spark RDDs to compute the total average Euclidean distance of the processed data set.

Although the use of Spark is optimal since it avoids storing the data on a hard disk, the calculation of the average of the Euclidean distances of the data set is costly. It is, therefore, essential to optimize this calculation. For this purpose two proposals have been made.

- **The First Solution**: This solution involves calculating the average of the Euclidean distances of a sample instead of processing the entire data set using the Map and Reduce functions. Since the computation of Euclidean distances requires fixing at each iteration a data object and computing the two-by-two Euclidean distance between this object and the rest of the data objects. Consequently, the number of iterations depends on the size of the processed data set. Therefore, the number of iterations is reduced compared to the processing of the whole data set.

- **The Second Solution**: The second solution consists of first reordering the whole data set randomly and then applying the MapPartitions function, which allows decomposing the data set into a specific number of partitions. Then, in each partition, the process starts by performing the computation of the two-by-two Euclidean distances of a part of the data. Then, the distances computed are summed and the partial averages of the Euclidean distances are calculated. Thus, the result produced by the MapPartitions function is transmitted to the Reducer, allowing the calculation of the total average of the Euclidean distances. In this solution, the MapPartitions and Reduce functions are invoked only once, so the calculation time is much reduced compared to the solution explained previously. Moreover, since this proposal requires the random reordering of data objects, the result given at each execution is different. In other words, using this solution, the algorithm can be executed several times and the best solution is saved. Figure 5.4 illustrates the flow of Spark RDD to calculate the total average of the Euclidean distances of the processed data set.

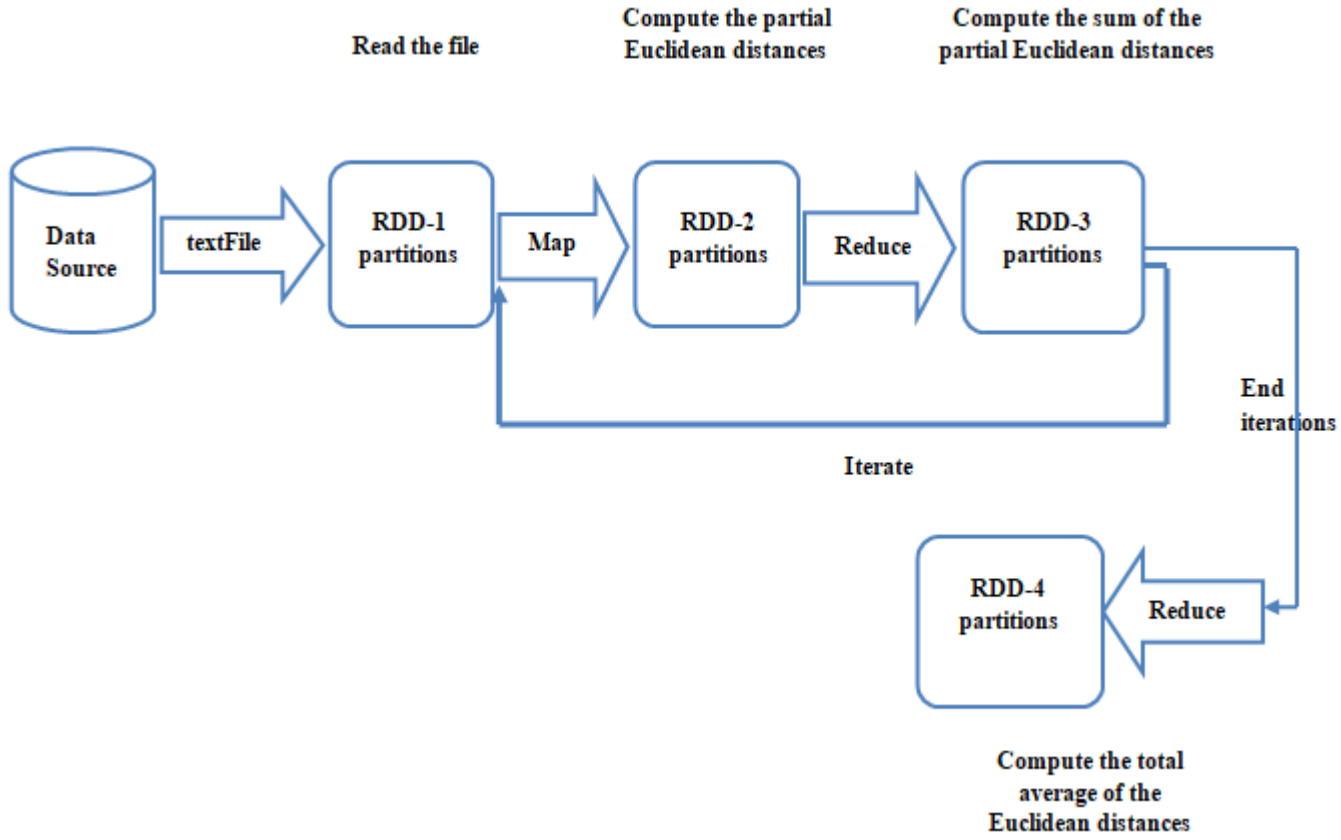

Figure 5.3: The parallel process of the average of Euclidean distance calculation

5.3.0.2 The construction of the initial clusters

This step starts by choosing the first data object in the data set as the first cluster center. Then, the data objects similar to this cluster center are gathered in such a way that the Euclidean distance between this cluster center and each data object in the whole data set is less than the total average of the Euclidean distances calculated in the previous step. Contrary to the sequential version of the algorithm in which the construction of the first cluster is realized in n iterations, the construction of the first cluster in the parallel version of the algorithm is done using the filter function, which selects the elements similar to the cluster center by calculating the Euclidean distances in parallel. Subsequently, the cluster center of the second cluster is determined in such a way that the data object similar to the first cluster center whose Euclidean distance is the greatest is compared to the data objects not similar to the center, so the data object whose

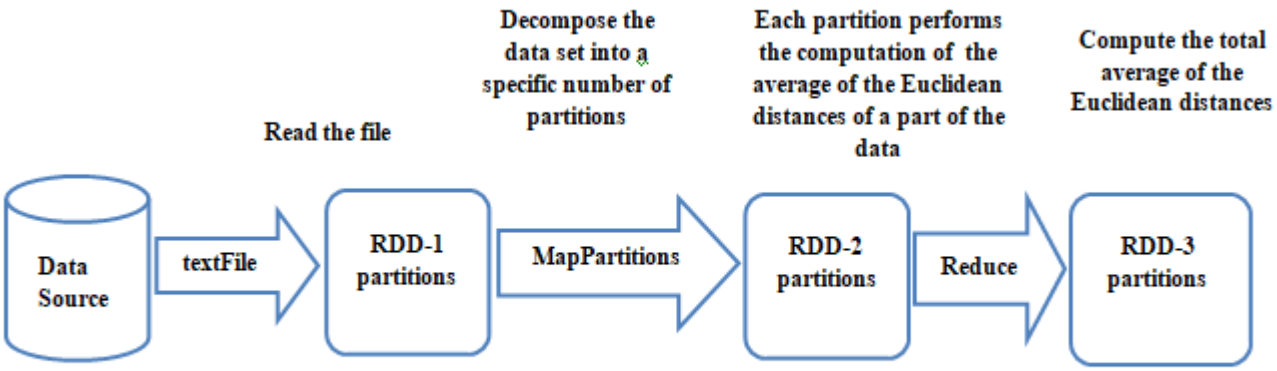

Figure 5.4: The parallel process of the average of Euclidean distance calculation (the second solution)

Euclidean distance yields an average value is chosen as the second cluster center. Consequently, the determination of the cluster center is realized using the filter function, which allows selecting the cluster center among the objects not similar to the first cluster center by calculating the Euclidean distances in parallel. In this way, the other cluster centers are determined successively to follow the same process. This process is finished when all the data objects are classified and there are no more data objects to choose from as a new cluster center. And lastly, it is important to mention that during the construction of the clusters, each data object can be placed in several clusters, and therefore the clusters achieved at the end of this step are overlapped. This allows for the enlargement of the size of the clusters and, accordingly, improves the quality of the cluster centers obtained. Fig.5.5 shows the cluster construction process using the Spark RDDs functions.

5.3.0.3 Assigning the data objects to the appropriate clusters and updating cluster centers

After having built the initial clusters in the previous step, this step consists of assigning to each cluster center found in the previous step the data objects similar to it by using the filter function. This function selects the data objects by performing the calculations of the Euclidean distances in parallel. Then, the Reduce function is applied to the RDD returned by the Filter function to update the value of the cluster center by computing the average of the data objects

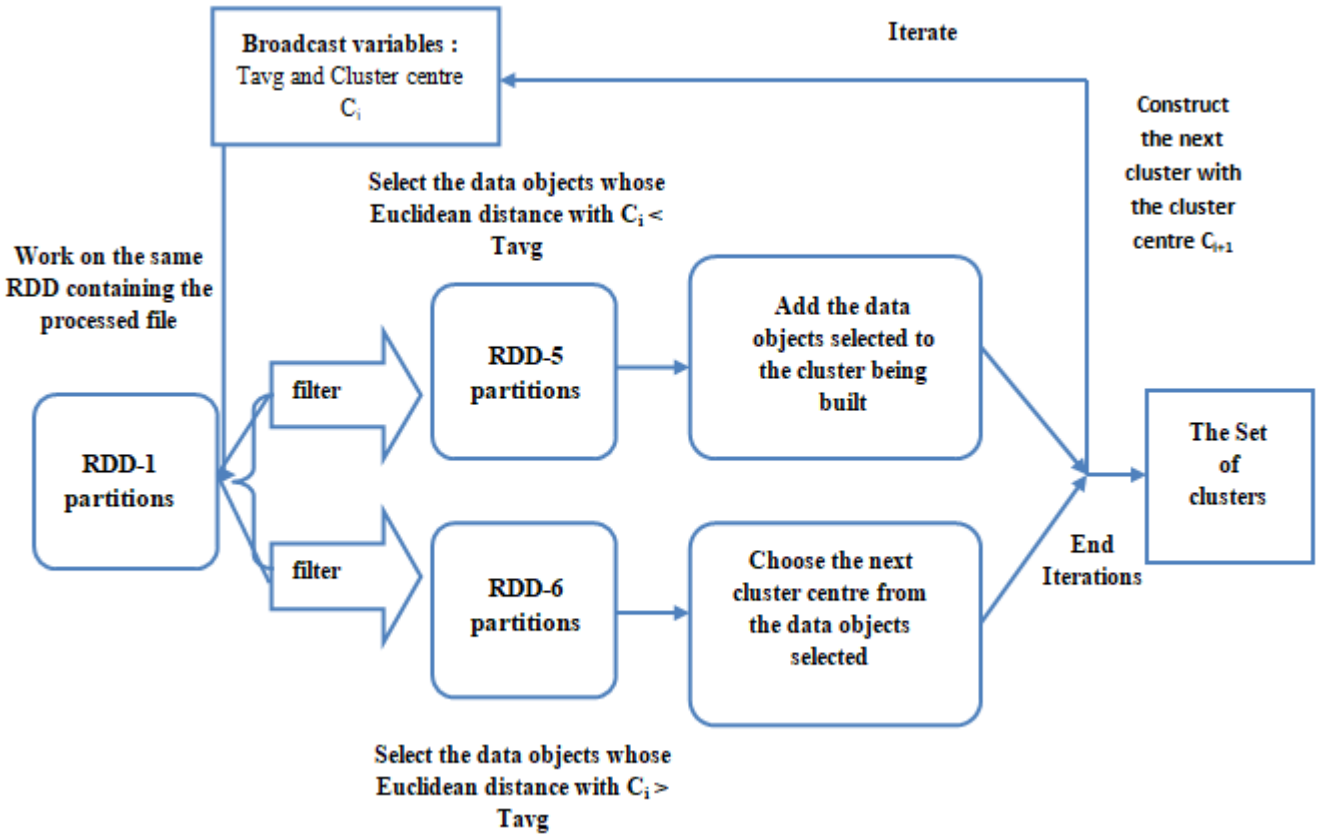

Figure 5.5: The parallel process of initial cluster construction

contained in the corresponding cluster. In this step, each data object can be associated with only one cluster, and consequently, there will be no overlaps in the resultant clusters. Fig.5.6 describes the process of obtaining the final hard clusters with the new cluster centers using the Filter and Reduce functions. Once the hard clusters with updated cluster centers and the number of cluster centers detected automatically have been obtained, the next step is to apply the parallel K-means provided by Spark mLlib with the input parameters obtained from the first stage, which are the cluster centers and the number of clusters.

5.3.1 The proposed parallel algorithm

The following pseudo-code (Algorithm 7) describes the stages of the parallel implementation of the PFK-means algorithm. Hence, the process begins by reading the file containing the set of quantitative data (lines 1 and 2). Subsequently, it seeks to calculate the average of the Euclidean

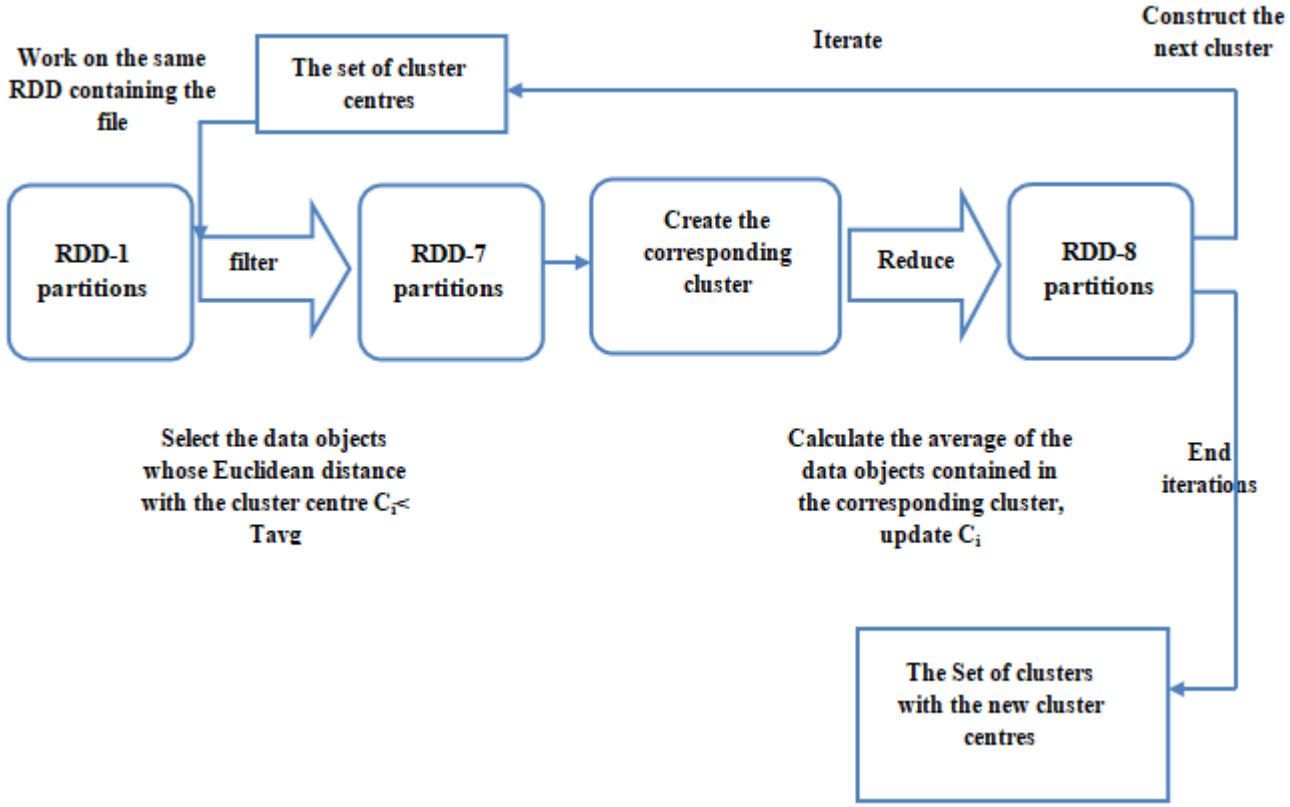

Figure 5.6: The parallel process of updating clusters

distances of the data set using one of the previously explained methods (line 3). The next step is to build the initial clusters to obtain the list of cluster centers to be used in the next step (from 4 to 17). Afterward, the process aims to assign the data objects to the appropriate centers and update the different centers (from 18 to 24). The last step intends to invoke the parallel K-means provided by Spark mLlib using the parameters obtained from the previous step (from 25 to 27).

5.4 Results and discussion

5.4.1 Experimental Setup

For the purpose of implementing the parallel version of PFK-means with Spark RDDs in stand-alone mode, a machine of 12GB memory and 1TB hard disk was configured with Spark (version 3.0.3) and Hadoop (2.7 version) and PyCharm IDE by installing useful Pyspark libraries (PySpark version is 3.9). Subsequently, the proposed algorithm was evaluated on real-world data sets

Algorithm 7 The Parallel implementation of PFK-means

Input:A set of n data objects X
Output:The initial cluster centers Tc_{next}. The number of clusters automatically computed
begin

ReadFile = sc.textFile("the file path")
dtSet = ReadFile.map(lines).cache()
TotalAverage = ComputeTotalAverage()
sc.broadcast($TotalAverage$)
$NextCenter = newList[0], Next = True$
while $Next$ **do**
    sc.broadcast($NextCenter$)
    Cluster = dtSet.filter($dist(X, NextCenter) < TotalAverage$)
    NextC= dtSet.filter($dist(X, NextCenter) > TotalAverage$)
    SortedList = NextC.sortBy($dist(X, FarthestElement)$)
    $NextCenter = MeanValue(SortedList.collect())$
    Add the constructed Cluster to Tc_{next}
    if $NextCenter == Null$ **then**
        $Next == False$
    end if
end while
Save the set of centers in $Centers$
$i \leftarrow 0$
while $i < ClustersSize$ **do**
    Cluster = dtSet.filter($dist(X, Centers[i]) < TotalAverage$)
    UpdateCenter=Cluster.reduce($computeMean(X, Y)$)
    Remove the filtred elements from the dataset dtSet
    Save the new cluster center in $NewCenters$
end while
$t_{max} = maxIterations, dt = dtSet, k = NewCentersSize$
$initializationMode = KMeansModel(NewCenters)$
FinalC=KMeans.train($dtSet, k, t_{max}, initializationMode$)
end begin

extracted from the UCI machine learning repository based on the sum of squared errors and the execution time to measure the clustering performance. In addition, the parallel PFK-means were compared with the parallel K-means provided by the mLlib library to demonstrate the efficiency of the developed solution. It is also crucial to notice that the parallel implementation of PFK-means has been tested locally in the PyCharm IDE and, therefore, the size of the data sets has been limited.

Table 5.1: Description of the real-life data sets.

data set	Data size	Number of Instances	Attributes	Cluster number
Soybean-small	3.39 Ko	47	35	4
Iris	4.58 Ko	150	4	3
Seeds	9.27 Ko	210	7	3
Heart	10.6 Ko	178	13	2
Wine	10.7 Ko	178	13	3
Pima Indian Diabetes	23.4 Ko	768	8	2
Letter-Recognition(LR)	0.715 Mo	20000	16	26
Satellite	0.750 Mo	6435	36	7

5.4.2 data sets description

Table 5.1 provides a short description of eight real-world datasets, extracted from the UCI machine learning repository [12] and Kaggle website, serving to validate the efficiency of the parallel implementation of PFK-means, including Soybean-small, Iris, Seeds, Heart, Wine, Pima Indian Diabetes, Letter Recognition (LR), and Satellite dataset. In this way, each data set is represented by a data size, number of instances, a determined number of clusters, and the number of attributes that constitutes the vectors of the data sets.

5.4.3 Results achieved in terms of the sum of squared errors

One of the effective metrics to evaluate the quality of clustering is the sum of the squared errors. This measure is obtained by calculating the sum of the Euclidean distances between each cluster center and the set of data objects belonging to this cluster. Therefore, the sum of the results found corresponds to the sum of squared errors of the set of clusters. Accordingly, the more minimal this measure is, the better the result is achieved.

Table 5.2 reports the sum of squared errors of the clustering result obtained by running the parallel PFK-means and the parallel K-means provided by Spark mLlib on the Letter Recognition (LR) data set. The execution of parallel PFK-means performing clustering by automatically detecting the number of clusters that can be varied by changing the position of the new cluster center determined at each iteration of the cluster construction step of the algorithm. Thus, the reported results are obtained by iterating the parallel K-means one, three, and ten times, respectively. In other words, for the parallel PFK-means, the reiteration is applied just to the

Table 5.2: The sum of squared errors of the clustering results on Letter-Recognition data set.

number of clusters	parallel PFK-means	parallel K-means (mLlib)	number of iterations
24	**117015.22**	118400.60	1
24	**113035.25**	113995.74	3
24	111162.61	**110717.19**	10
26	**114813.58**	118262.82	1
26	**111198.35**	112354.80	3
26	**109506.13**	109853.67	10

second phase of the algorithm, which consists of applying the parallel K-means implementation of Spark mLlib using the cluster centers and the number of clusters found in the first phase of the algorithm. In this respect, the acquired results are compared with the parallel K-means of Spark mLlib using the random initialization mode.

By observing the results displayed in the table 5.2, it can be seen that the parallel PFK-means outperforms the parallel K-means in terms of the sum of squared errors, except for the tenth iteration with k = 24, for which the parallel K-means exceeds the proposed algorithm. It is also noteworthy that the PFK-means produces good results from the first iteration, implying that the cluster centers discovered in the first phase of the algorithm are well-positioned in comparison to the cluster centers discovered using the random initialization mode. Besides the fact that the parallel PFK-means allows discovering the number of clusters automatically, the result given by the developed algorithm is stable.

Table 5.3 presents a comparison of the parallel PFK-means and the parallel K-means of Spark mLlib using the random initialization mode in terms of the sum of squared errors for the real-world data sets Soybean, Iris, Seeds, Heart, Wine, Pima, and Satellite. The results clearly show that the parallel PFK-means exceeds the parallel K-means in terms of the sum of squared errors. In addition, the parallel PFK-means detects automatically the number of clusters contrary to the parallel K-means that requires fixing the number of clusters. It is also important to note that the results found by the parallel version of PFK-means is stable.

Table 5.3: The sum of squared errors of the clustering results on real-world data sets.

data set	parallel PFK-means	parallel K-means (mLlib)
Soybean	**97.48**	103.61
Iris	**97.34**	97.70
Seeds	**322.74**	326.16
Wine	**18889.14**	19015.23
Heart	**23800.17**	25599.19
Pima	**49403.81**	64315.08
Satellite	**301200.24**	378609.15

5.4.4 Results of the processing time

In order to evaluate the performance of the parallel PFK-means concerning running time, the second phase of the proposed algorithm has been compared with the parallel K-means of Spark mLlib using the random initialization mode, which allows to randomly generate the cluster centers. In this case, the number of clusters k used is the number detected by the first phase of the parallel PFK-means, and therefore the number of cluster centers must be fixed before the execution. On the other hand, by running the second phase of the PFK-means, the number of clusters is automatically detected and assigned to the parallel K-means as well as the set of cluster centers.

Figure 5.7 illustrates the execution time of the second stage of the parallel PFK-means compared to the execution time of the parallel K-means on Soybean, Iris, Seeds, Heart, Wine, Pima, Letter-Recognition, and Satellite data sets. Figure 5.7 shows that the PFK-means outperforms the parallel K-means for all the data sets used.

The main steps of the parallel version of PFK-means include the initial cluster construction step, assigning data objects and updating clusters, and the execution of the parallel K-means of spark mLlib. The second experiment consists in comparing the execution time of these three principal steps using the different real-world data sets. These data sets have been categorized according to their dimensions and, more precisely, according to the number of attributes that characterize each data set. Thus, figure 5.8 depicts the execution time of the major steps of the proposed algorithm for Iris, Seeds, Heart, Wine, and Pima. Figure 5.9 illustrates the result of execution time of the three principal steps for Soybean, Letter, and Satellite data sets having

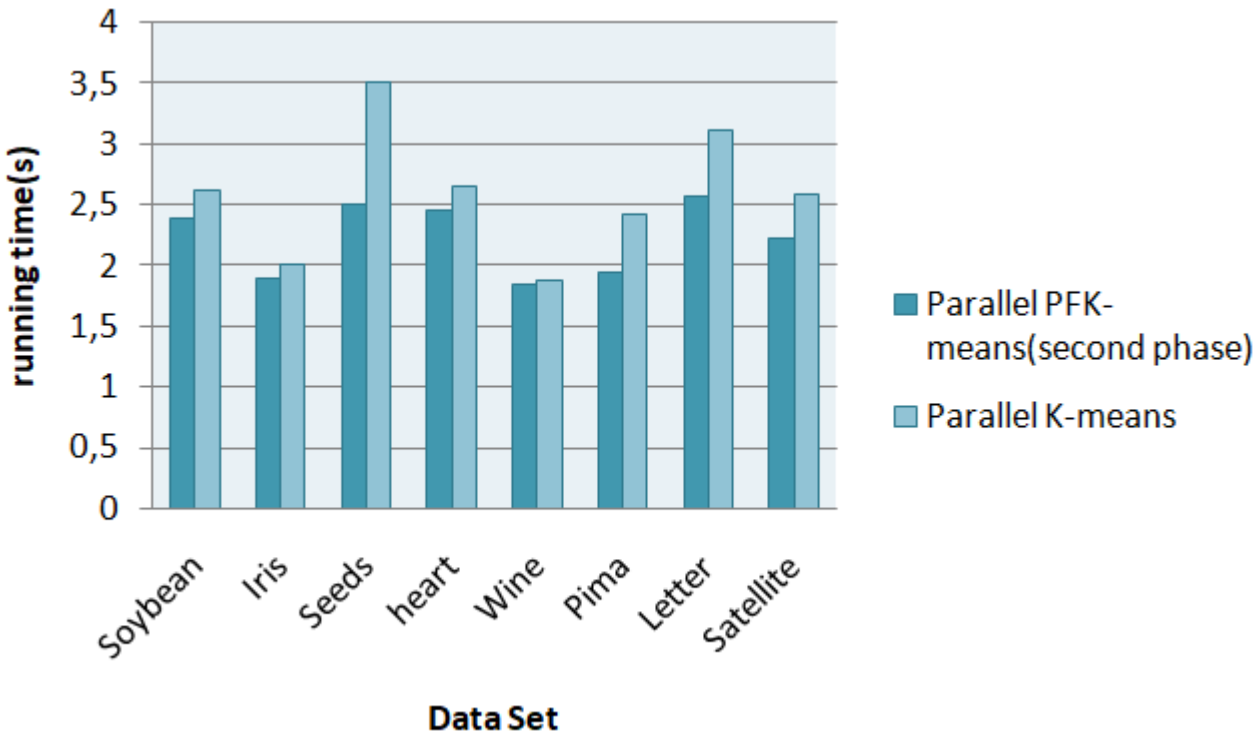

Figure 5.7: Clustering time of the second stage of parallel PFK-means and parallel K-means.

very high dimensions.

According to these two figures, the step that takes more time is the step of initial cluster construction. On the other hand, the time consumed by the execution of the parallel k-means is negligible compared to the time consumed in the first step and the step of attribution of the data objects and updating the set of clusters. The high execution time of the initial cluster construction step is explained by the fact that this step includes the discovery of the initial cluster centers and the construction of the set of initial clusters, which are done in several iterations that depend on the number of clusters automatically detected by the algorithm. Thus, the greater the number of clusters, the greater the time consumed. In addition, the dimension of the data set is another factor that is considered, because the more attributes there are, the greater the computations performed in the cluster construction step.

From the results shown in figure 5.9, it is obvious that the execution time of the three main steps of the algorithm is the largest for the Letter data set since, in addition to its high dimension, the number of classes discovered is also very large (k=26). Moreover, Soybean and Satellite

data sets also have larger dimensions and, therefore, a larger execution time compared to the execution time of the data sets presented in figure 5.9. As previously explained, the proposed

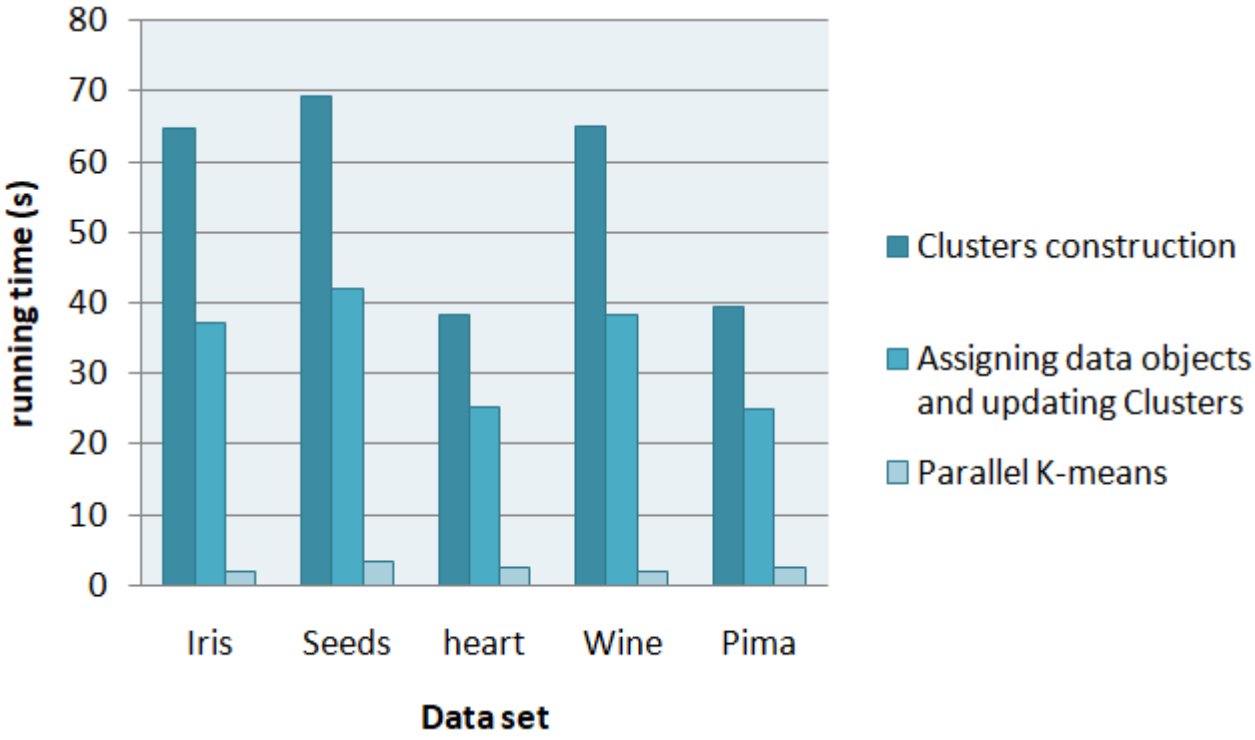

Figure 5.8: Clustering time on Iris, Seeds, heart, Wine, and Pima data sets.

algorithm includes the step of building the initial cluster centers as well as updating the set of obtained clusters. These steps are mainly based on Spark RDD functions: Filter and Reduce. Figures 5.10 and 5.7 show the execution time of the Filter and Reduce operations of the two main steps of the first phase of the proposed parallel algorithm. The Filter and Reduce operations of the global average of the Euclidean distance step are not considered in this experience since this step is independent of the cluster construction process. Obviously, the execution time of the Filter operations is more costly than that of the Reduce operations for all the real world data sets used. This is due to the fact that the majority of the operations performed for the construction of the initial centers are performed by the Filter function, while the Reduce function is used only for the calculation of the new centers in the cluster update step. Another factor that influences the obtained result is the fact that the algorithm is executed in local mode. Thus, the result can be different in the case of execution of the algorithm in a cluster by distributing the different tasks

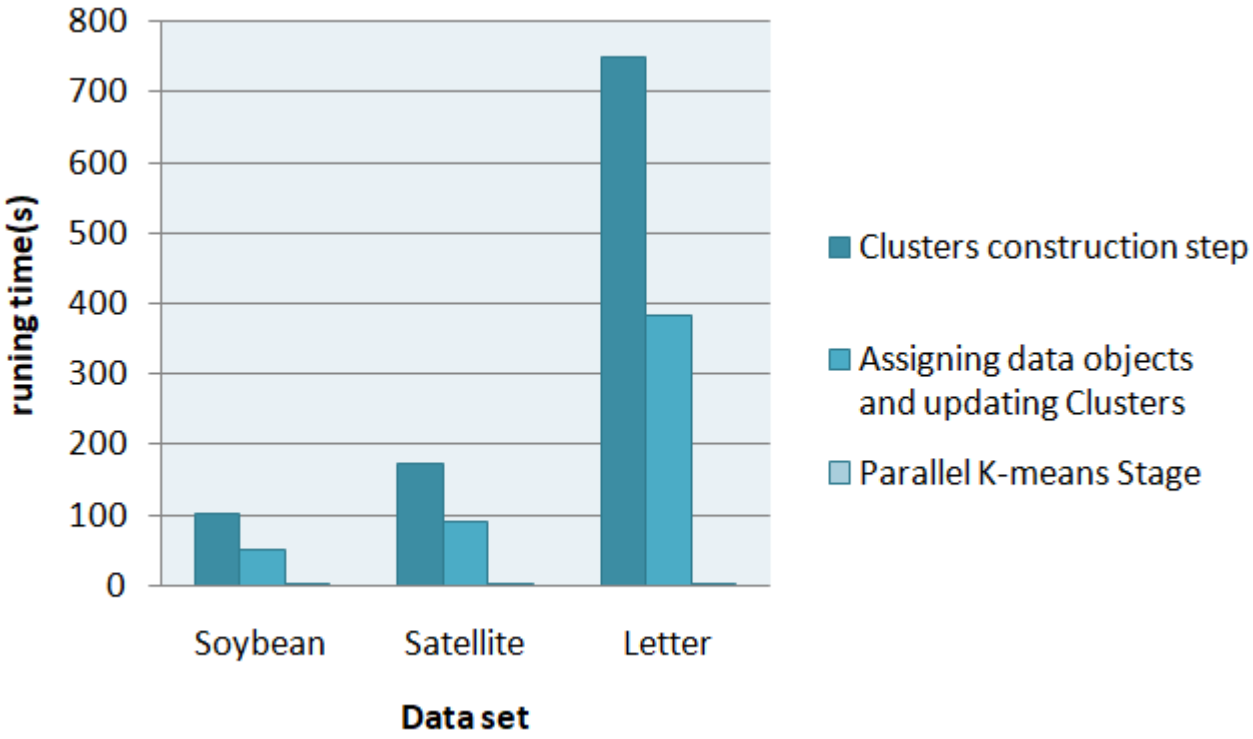

Figure 5.9: Clustering time on Soybean, Satellite, and Letter data sets.

of the algorithm on several nodes.

5.4.5 Comparison between the sequential version of PFK-means and the parallel version

With the purpose of proving the efficiency of the parallel implementation of the enhanced version of PFK-means over the sequential version, a comparison was made between these two versions based on the sum of squared errors on the Soybean, Iris, Seeds, Wine, Pima, and Letter Recognition data. Table 5.4 illustrates the results obtained, which clearly show that the parallel implementation of the parallel version of PFK-means outperforms the serial version in terms of the sum of squared errors and therefore the parallel implementation enhances the sequential version of PFK-means.

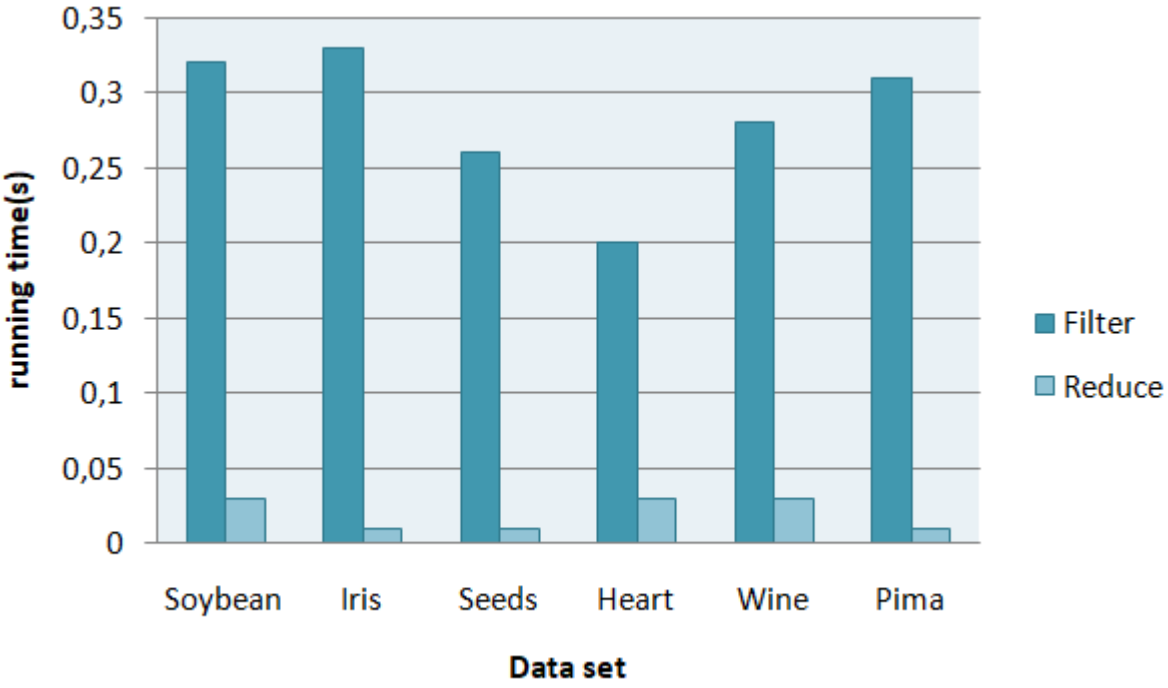

Figure 5.10: Clustering time of Filter and Reduce operations.

Table 5.4: The sum of squared errors of the clustering results on real-world data sets.

data set	parallel PFK-means	serial PFK-means
Soybean	**97.48**	207.96
Iris	97.34	**78.94**
Seeds	**322.74**	587.31
Wine	**1.8E+04**	2.63E+06
Pima	**4.9E+04**	5.18E+06
Letter Recognition	**1.09E+05**	6.16E+05

5.4.6 Discussion of the obtained results

According to the experiments that were conducted on real-world data sets, it was proved that the proposed algorithm is a parameter-free clustering algorithm since it can automatically determine the set of cluster centers and the number of clusters without specifying any input parameters, and therefore the suggested solution is suitable for unsupervised learning. However, running the parallel version of k-means provided by Spark mLlib requires fixing the number of clusters as well as specifying the initial cluster centers. Moreover, the execution of the developed algorithm using real-world data sets gives significant results compared to the results achieved by the

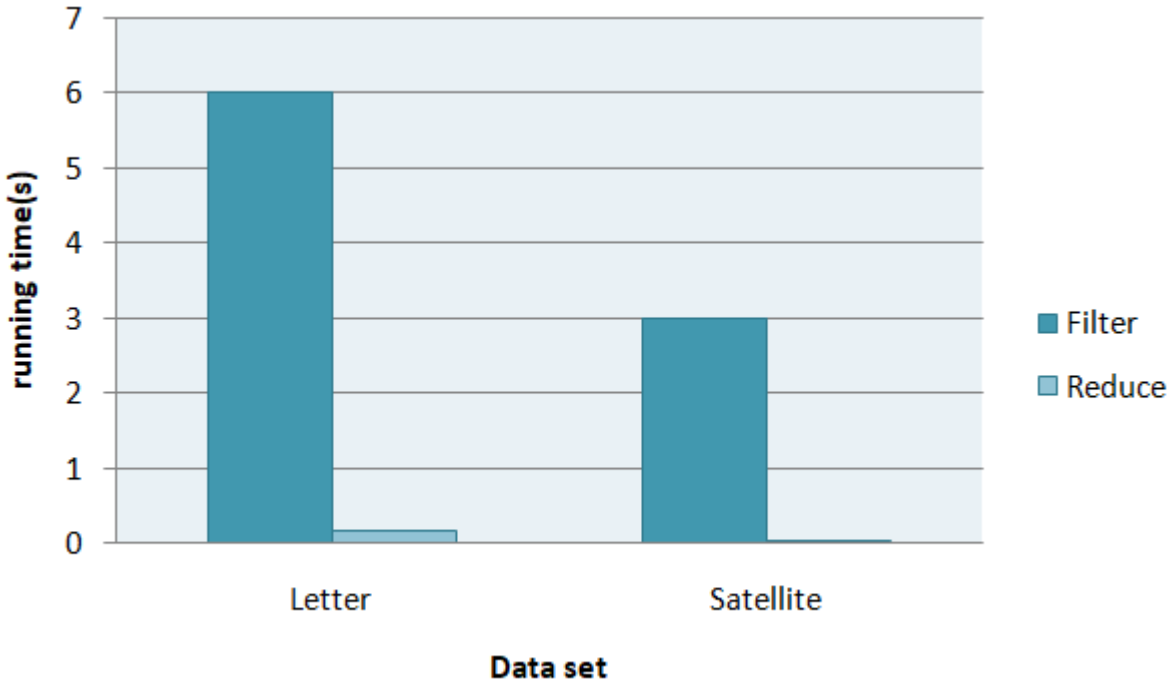

Figure 5.11: Clustering time on Soybean, Satellite, and Letter data sets.

parallel implementation of K-means provided by Spark mLlib in terms of the sum of squared errors and execution time. In addition to that, the parallel implementation of the method allows the distribution of the tasks constituting the main steps of the algorithm on several nodes, which allows for the processing of very large files.

5.5 Conclusion

The main concern of this chapter is to design a new parallel clustering algorithm based on the main steps of the PFK-means clustering algorithm for processing large quantitative data sets by bringing a significant improvement in the way of determining the initial cluster centers in the cluster construction step. Thus, this chapter exposed the design of the proposed parallel algorithm and a detailed explanation of the main steps. Furthermore, experiments based on UCI data sets were conducted based on the sum of squared errors and the execution time. The results have clearly shown that the suggested parallel algorithm is an efficient clustering algorithm as it can automatically construct the initial cluster centers as well as the number of clusters

without having to specify any parameters beforehand. Besides, it has been demonstrated that the developed algorithm outperforms the parallel K-means provided by Spark mLlib in terms of squared errors and processing time.

CHAPTER 6

GENERAL CONCLUSION AND PROSPECTS

This chapter summarizes the concepts discussed in this thesis as well as the different contributions proposed. Thus, section 6.1 provides the general conclusion of the thesis. Section 6.2 outlines the different perspectives related to the work accomplished..

6.1 Conclusion

The emergence of new concepts such as Big Data and artificial intelligence has given rise to new challenges related to the complex types of data generated and the lack of sophisticated tools and techniques to keep up with this unprecedented development. It is, therefore, necessary to unify efforts to develop efficient solutions to better exploit this data and extract knowledge to support decision-making.

Clustering Big Data is a new technique exploiting the properties of clustering and the advantages of Big Data platforms to process an immense amount of data without having to define complex similarity measures or manually specified parameters.

The main objective of this thesis is to design and develop new heuristics for Big Data clustering based on two main approaches: relational analysis and the implementation of new Big Data platforms. Thus, the first contribution consists of an in-depth study of existing Big Data clustering algorithms categorized into two main categories: horizontal scaling platforms and vertical scaling

platforms. The first category includes peer-to-peer networks, MapReduce, and Spark platforms. The second category consists of Multi-core processors, Graphics Processing Unit, and Field Programmable Gate Arrays platforms. The reviewed algorithms are then compared according to the initial parameters, the type of data processed, and the criteria of clustering validation. The second contribution focuses on improving the $Transitives$ heuristic for clustering categorical data by taking advantages of the Relational Analysis approach. This approach ensures that the maximum number of clusters is automatically detected without having to specify complex parameters. The improvements brought to this new heuristic generally concern two aspects. The first one concerns the way of calculating the representative of a constructed cluster. While the second aspect focuses on the manner to process data objects. The third contribution consists of combining the E-Transitive heuristic and the traditional k-means to solve the issue of the determination of the initialization parameters of k-means. The process aims to develop a new algorithm designed for clustering quantitative data and its different variants concerning overlapping clustering or hard clustering. Despite the beneficial properties of the parameter-free clustering algorithm and its ability to automatically detect the number of centers as well as the initial centers, the problem of handling big data is still present. One effective solution to improve the performance of PFK-means is to exploit the spark framework. The last contribution aims at parallelizing the main tasks of the PFK-means algorithm by improving the approach to determining the initial centers in the construction stage. Thus, it proceeds by applying successive RDD functions and implementing the parallel k-means provided by the mLlib library.

6.2 Prospects

- We have proposed an enhanced version of the Transitive heuristic for processing categorical data based on the Relational Analysis, it will be beneficial to develop the parallel version of the improved version using Big Data platforms to allow processing of increasingly large data sets. In addition, it will be important to process heterogeneous data types to address new issues and challenges.

- We have developed the parallel version of the parameter-free clustering algorithm based on k-means using the stand-alone mode. In our future research, we intend to concentrate on creating a remote server configuration to distribute the different tasks of the algorithm on multiple nodes and therefore process very large files. In addition, we will establish a deep analysis to compare both the stand-alone mode implementation and cluster mode implementation. Furthermore, we will focus on the implementation of the proposed algorithm using other similarity measures and various data sets. It may also be possible to apply the proposed solution to perform real-time processing.

- Hybrid parallel programming is a promising way to take advantage of both shared memory and distributed memory systems. It would be interesting to investigate other combinations of the available Big Data platforms and study if they can lead to further scalability improvements.

- Despite the performance of parallel clustering algorithms developed in different areas using a variety of frameworks and architectures, the techniques and tools offered by these algorithms are sometimes unable to deal with all the problems related to Big Data and therefore cover all the fields using Big Data processing. The current trend for clustering is to combine it with other approaches such as classification, machine learning, etc.

BIBLIOGRAPHY

[1] A. ADINETZ, J. KRAUS, J. MEINKE, AND D. PLEITER, *Gpumafia: Efficient subspace clustering with mafia on gpus*, in European Conference on Parallel Processing, Springer, 2013, pp. 838–849.

[2] C. C. AGGARWAL, J. HAN, J. WANG, AND P. S. YU, *A framework for clustering evolving data streams*, in Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29, VLDB '03, Berlin, Germany, 2003, VLDB Endowment, pp. 81–92.

[3] J. AH-PINE AND J.-F. MARCOTORCHINO, *Overview of the relational analysis approach in data-mining and multi-criteria decision making*, in Web intelligence and intelligent agents, IntechOpen, 2010.

[4] M. AHMED, R. SERAJ, AND S. M. S. ISLAM, *The k-means algorithm: A comprehensive survey and performance evaluation*, Electronics, 9 (2020), p. 1295.

[5] S. AKHTER AND J. ROBERTS, *Multi-core programming: increasing performance through software multi-threading*, Books by engineers, for engineers, Intel Press, Hillsboro, Or, 1st ed., 2006.

[6] M. C. ALTINIGNELI, C. PLANT, AND C. BÖHM, *Massively parallel expectation maximization using graphics processing units*, in Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, Chicago, Illinois, USA, 2013, ACM, pp. 838–846.

[7] F. AN, T. KOIDE, AND H. J. MATTAUSCH, *A k-means-based multi-prototype high-speed learning system with fpga-implemented coprocessor for 1-nn searching*, IEICE TRANSACTIONS on Information and Systems, E95-D (2012), pp. 2327–2338.

[8] G. ANDRADE, G. RAMOS, D. MADEIRA, R. SACHETTO, R. FERREIRA, AND L. ROCHA, *G-dbscan: A gpu accelerated algorithm for density-based clustering*, Procedia Computer Science, 18 (2013), pp. 369–378.

[9] M. ANKERST, M. M. BREUNIG, H.-P. KRIEGEL, AND J. SANDER, *Optics: Ordering points to identify the clustering structure*, in Proceedings of the 1999 ACM SIGMOD International

Conference on Management of Data, SIGMOD '99, Philadelphia, Pennsylvania, USA, 1999, ACM, pp. 49–60.

[10] D. ARTHUR AND S. VASSILVITSKII, *k-means++: The advantages of careful seeding*, in Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[11] A. ASHABI, S. B. SAHIBUDDIN, AND M. SALKHORDEH HAGHIGHI, *The systematic review of k-means clustering algorithm*, in 2020 The 9th International Conference on Networks, Communication and Computing, 2020, pp. 13–18.

[12] A. ASUNCION AND D. NEWMAN, *Uci machine learning repository*, 2007.

[13] R. AZIMI, H. SAJEDI, AND M. GHAYEKHLOO, *A distributed data clustering algorithm in p2p networks*, Applied Soft Computing, 51 (2017), pp. 147–167.

[14] A. M. BAGIROV, *Modified global k-means algorithm for minimum sum-of-squares clustering problems*, Pattern Recognition, 41 (2008), pp. 3192–3199.

[15] B. BALUSAMY, S. KADRY, A. H. GANDOMI, ET AL., *Big Data: Concepts, Technology, and Architecture*, John Wiley & Sons, 2021.

[16] A. BANHARNSAKUN, *A mapreduce-based artificial bee colony for large-scale data clustering*, Pattern Recognition Letters, 93 (2017), pp. 78–84.

[17] N. BANSAL, A. BLUM, AND S. CHAWLA, *Correlation clustering*, Machine learning, 56 (2004), pp. 89–113.

[18] T. BEIER, F. A. HAMPRECHT, AND J. H. KAPPES, *Fusion moves for correlation clustering*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3507–3516.

[19] T. BEIER, T. KROEGER, J. H. KAPPES, U. KOTHE, AND F. A. HAMPRECHT, *Cut, glue & cut: A fast, approximate solver for multicut partitioning*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 73–80.

[20] A. BEN-DOR, R. SHAMIR, AND Z. YAKHINI, *Clustering gene expression patterns*, Journal of Computational Biology, 6 (1999), pp. 281–297.

[21] P. BERKHIN, *A survey of clustering data mining techniques*, in Grouping multidimensional data, Springer, 2006, pp. 25–71.

[22] N. BHARILL, A. TIWARI, AND A. MALVIYA, *Fuzzy based scalable clustering algorithms for handling big data using apache spark*, IEEE Transactions on Big Data, 2 (2016), pp. 339–352.

[23] S. BOUHOUT, Y. OUBENAALLA, AND E. H. NFAOUI, *Comparative study of two parallel algorithm k-means and dbscan clustering on spark platform*, in International Conference on Advanced Intelligent Systems for Sustainable Development, Springer, 2020, pp. 245–262.

[24] S. D. BROWN, R. J. FRANCIS, J. ROSE, AND Z. G. VRANESIC, *Field-Programmable Gate Arrays*, The Kluwer International Series in Engineering and Computer Science, Springer US, Boston, MA, 1992.

[25] A. BUSTAMAM, K. BURRAGE, AND N. A. HAMILTON, *Fast parallel markov clustering in bioinformatics using massively parallel computing on gpu with cuda and ellpack-r sparse format*, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 9 (2012), pp. 679–692.

[26] P. CARBONE, A. KATSIFODIMOS, S. EWEN, V. MARKL, S. HARIDI, AND K. TZOUMAS, *Apache flink: Stream and batch processing in a single engine*, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 36 (2015).

[27] J. L. CARBONERA AND M. ABEL, *Cbk-modes: A correlation-based algorithm for categorical data clustering.*, in ICEIS (1), 2015, pp. 603–608.

[28] J. M. CEBRIAN, B. IMBERNÓN, J. SOTO, J. M. GARCÍA, AND J. M. CECILIA, *High-throughput fuzzy clustering on heterogeneous architectures*, Future Generation Computer Systems, (2020).

[29] M. E. CELEBI, H. A. KINGRAVI, AND P. A. VELA, *A comparative study of efficient initialization methods for the k-means clustering algorithm*, Expert systems with applications, 40 (2013), pp. 200–210.

[30] C. CHEN, K. LI, A. OUYANG, AND K. LI, *Flinkcl: an opencl-based in-memory computing architecture on heterogeneous cpu-gpu clusters for big data*, IEEE Transactions on Computers, 67 (2018), pp. 1765–1779.

[31] C. CHEN, K. LI, A. OUYANG, Z. ZENG, AND K. LI, *Gflink: An in-memory computing architecture on heterogeneous cpu-gpu clusters for big data*, IEEE Transactions on Parallel and Distributed Systems, 29 (2018), pp. 1275–1288.

[32] K. CHOWDHURY, D. CHAUDHURI, AND A. K. PAL, *An entropy-based initialization method of k-means clustering on the optimal number of clusters*, Neural Computing and Applications, (2020), pp. 1–18.

[33] I. CORDOVA AND T.-S. MOH, *Dbscan on resilient distributed datasets*, in International Conference on High Performance Computing Simulation (HPCS), Amsterdam, Netherlands, 2015, IEEE, pp. 531–540.

[34] X. CUI, J. GAO, AND T. E. POTOK, *A flocking based algorithm for document clustering analysis*, Journal of Systems Architecture, 52 (2006), pp. 505–515.

[35] X. CUI, P. ZHU, X. YANG, K. LI, AND C. JI, *Optimized big data k-means clustering using mapreduce*, The Journal of Supercomputing, 70 (2014), pp. 1249–1259.

[36] S. CUOMO, V. DE ANGELIS, G. FARINA, L. MARCELLINO, AND G. TORALDO, *A gpu-accelerated parallel k-means algorithm*, Computers & Electrical Engineering, (2017).

[37] Z. DAFIR, Y. LAMARI, AND S. C. SLAOUI, *A survey on parallel clustering algorithms for big data*, Artificial Intelligence Review, (2020), pp. 1–33.

[38] Z. DAFIR AND S. SLAOUI, *An efficient parallel algorithm for clustering big data based on the spark framework*.

[39] ——, *Achievements of artificial intelligence in the past and during the covid-19 era to tackle deadly diseases*, in International Symposium on Automation, Mechanical and Design Engineering, Springer, 2023, pp. 181–191.

[40] J. DEAN AND S. GHEMAWAT, *Mapreduce: simplified data processing on large clusters*, in Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6, OSDI'04, Berkeley, CA, USA, 2004, USENIX Association.

[41] Z. DENG, Y. HU, M. ZHU, X. HUANG, AND B. DU, *A scalable and fast optics for clustering trajectory big data*, Cluster Computing, 18 (2015), pp. 549–562.

[42] J. EKANAYAKE, H. LI, B. ZHANG, T. GUNARATHNE, S.-H. BAE, J. QIU, AND G. FOX, *Twister: a runtime for iterative mapreduce*, in Proceedings of the 19th ACM international symposium on high performance distributed computing, ACM, 2010, pp. 810–818.

[43] A. J. ENRIGHT, S. VAN DONGEN, AND C. A. OUZOUNIS, *An efficient algorithm for large-scale detection of protein families*, Nucleic Acids Research, 30 (2002), pp. 1575–1584.

[44] A. ERDEM AND T. İ. GÜNDEM, *M-fdbscan: A multicore density-based uncertain data clustering algorithm*, TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES, 22 (2014), pp. 143–154.

[45] M. ESTER, H.-P. KRIEGEL, J. SANDER, AND X. XU, *A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise*, in Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD'96, Portland, Oregon, 1996, AAAI Press, pp. 226–231.

[46] U. FAROOQ, Z. MARRAKCHI, AND H. MEHREZ, *Fpga architectures: An overview*, in Tree-based Heterogeneous FPGA Architectures, Springer New York, New York, NY, 2012, ch. 2, pp. 7–48.

[47] P. FRÄNTI AND S. SIERANOJA, *K-means properties on six clustering benchmark datasets (2018)*, URL: http://cs. uef. fi/sipu/datasets.

[48] A. GATTAL, F. ABBAS, AND M. R. LAOUAR, *Automatic parameter tuning of k-means algorithm for document binarization*, in Proceedings of the 7th International Conference on Software Engineering and New Technologies, 2018, pp. 1–4.

[49] J. GEHWEILER AND H. MEYERHENKE, *A distributed diffusive heuristic for clustering a virtual p2p supercomputer*, in IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), Atlanta, GA, USA, 2010, IEEE, pp. 1–8.

[50] P. GEPNER AND M. F. KOWALIK, *Multi-core processors: New way to achieve high system performance*, in International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06), Bialystok, Poland, 2006, pp. 9–13.

[51] E. GHASEMI AND P. CHOW, *Accelerating apache spark with fpgas*, Concurrency and Computation: Practice and Experience, 31 (2019), p. e4222.

[52] F. GOUINEAU, T. LANDRY, AND T. TRIPLET, *Patchwork, a scalable density-grid clustering algorithm*, in Proceedings of the 31st Annual ACM Symposium on Applied Computing, SAC '16, Pisa, Italy, 2016, ACM, pp. 824–831.

[53] P. GOYAL, S. KUMARI, D. KUMAR, S. BALASUBRAMANIAM, AND N. GOYAL, *Parallelizing optics for multicore systems*, in Proceedings of the 7th ACM India Computing Conference, ACM, 2014, p. 17.

[54] A. HADIAN AND S. SHAHRIVARI, *High performance parallel k-means clustering for disk-resident datasets on multi-core cpus*, The Journal of Supercomputing, 69 (2014), pp. 845–863.

[55] M. HALL, E. FRANK, G. HOLMES, B. PFAHRINGER, P. REUTEMANN, AND I. H. WITTEN, *The weka data mining software: an update*, ACM SIGKDD explorations newsletter, 11 (2009), pp. 10–18.

[56] D. HAN, A. AGRAWAL, W.-K. LIAO, AND A. CHOUDHARY, *A novel scalable dbscan algorithm with spark*, in IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, IL, USA, 2016, IEEE, pp. 1393–1402.

[57] J. HAN, J. PEI, AND M. KAMBER, *Data mining: concepts and techniques*, Elsevier, 2011.

[58] P. HARISH AND P. J. NARAYANAN, *Accelerating large graph algorithms on the gpu using cuda*, in High Performance Computing – HiPC 2007, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2007, pp. 197–208.

[59] J. A. HARTIGAN AND M. A. WONG, *Algorithm as 136: A k-means clustering algorithm*, Applied Statistics, 28 (1979), p. 100.

[60] T. C. HAVENS, J. C. BEZDEK, C. LECKIE, L. O. HALL, AND M. PALANISWAMI, *Fuzzy c-means algorithms for very large data*, IEEE Transactions on Fuzzy Systems, 20 (2012), pp. 1130–1146.

[61] Y. HE, H. TAN, W. LUO, S. FENG, AND J. FAN, *Mr-dbscan: a scalable mapreduce-based dbscan algorithm for heavily skewed data*, Frontiers of Computer Science, 8 (2014), pp. 83–99.

[62] W. HENDRIX, M. M. A. PATWARY, A. AGRAWAL, W.-K. LIAO, AND A. CHOUDHARY, *Parallel hierarchical clustering on shared memory platforms*, in 2012 19th International Conference on High Performance Computing, IEEE, 2012, pp. 1–9.

[63] P. HUANG, X. LI, AND B. YUAN, *A parallel gpu-based approach to clustering very fast data streams*, in Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, CIKM '15, Melbourne, Australia, 2015, ACM, pp. 23–32.

[64] H. M. HUSSAIN, K. BENKRID, H. SEKER, AND A. T. ERDOGAN, *Fpga implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data*, in NASA/ESA Conference on Adaptive Hardware and Systems (AHS), San Diego, California, USA, 2011, IEEE, pp. 248–255.

[65] H. ISMKHAN, *Ik-means-+: An iterative clustering algorithm based on an enhanced version of the k-means*, Pattern Recognition, 79 (2018), pp. 402–413.

[66] A. K. JAIN, *Data clustering: 50 years beyond k-means*, Pattern recognition letters, 31 (2010), pp. 651–666.

[67] F. JIA, C. WANG, X. LI, AND X. ZHOU, *Sakma: Specialized fpga-based accelerator architecture for data-intensive k-means algorithms*, in Algorithms and Architectures for Parallel Processing, Zhangjiajie, China, 2015, Springer, Cham, pp. 106–119.

[68] C. JIN, M. MOSTOFA, A. PATWARY, A. AGRAWAL, W. HENDRIX, W.-K. LIAO, AND A. CHOUDHARY, *Disc: A distributed single-linkage hierarchical clustering algorithm using mapreduce*, in The 4th International SC Workshop on Data Intensive Computing in the Clouds (DataCloud, Citeseer, 2013.

[69] R. JIN, C. KOU, R. LIU, AND Y. LI, *Efficient parallel spectral clustering algorithm design for large data sets under cloud computing environment*, Journal of Cloud Computing: Advances, Systems and Applications, 2 (2013), p. 18.

[70] T. KANUNGO, D. M. MOUNT, N. S. NETANYAHU, C. D. PIATKO, R. SILVERMAN, AND A. Y. WU, *An efficient k-means clustering algorithm: analysis and implementation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 24 (2002), pp. 881–892.

[71] G. KARYPIS AND V. KUMAR, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392.

[72] B. W. KERNIGHAN AND S. LIN, *An efficient heuristic procedure for partitioning graphs*, The Bell system technical journal, 49 (1970), pp. 291–307.

[73] M. KEUPER, E. LEVINKOV, N. BONNEEL, G. LAVOUÉ, T. BROX, AND B. ANDRES, *Efficient decomposition of image and mesh graphs by lifted multicuts*, in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1751–1759.

[74] S. S. KHAN AND A. AHMAD, *Cluster center initialization algorithm for k-means clustering*, Pattern recognition letters, 25 (2004), pp. 1293–1302.

[75] J. M. KRAUS AND H. A. KESTLER, *A highly efficient multi-core algorithm for clustering extremely large datasets*, BMC Bioinformatics, 11 (2010), p. 169.

[76] H.-P. KRIEGEL AND M. PFEIFLE, *Density-based clustering of uncertain data*, in Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, Chicago, Illinois, USA, 2005, ACM, pp. 672–677.

[77] C. LANCZOS, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, United States Governm. Press Office Los Angeles, CA, 1950.

[78] R. LENZ, *Big data: Ethics and law*, Available at SSRN 3459004, (2019).

[79] E. LEVINKOV, A. KIRILLOV, AND B. ANDRES, *A comparative study of local search algorithms for correlation clustering*, in German Conference on Pattern Recognition, Springer, 2017, pp. 103–114.

[80] C. LI, Y. ZHANG, M. JIAO, AND G. YU, *Mux-kmeans: Multiplex kmeans for clustering large-scale data set*, in Proceedings of the 5th ACM Workshop on Scientific Cloud Computing, ScienceCloud '14, Vancouver, BC, Canada, 2014, ACM, pp. 25–32.

[81] A. LIKAS, N. VLASSIS, AND J. J. VERBEEK, *The global k-means clustering algorithm*, Pattern recognition, 36 (2003), pp. 451–461.

[82] F. LIN AND W. W. COHEN, *Power iteration clustering*, in Proceedings of the 27th international conference on machine learning (ICML-10), Haifa, Israel, 2010, Omnipress, pp. 655–662.

[83] K. W. LIN, C.-H. LIN, AND C.-Y. HSIAO, *A parallel and scalable cast-based clustering algorithm on gpu*, Soft Computing, 18 (2014), pp. 539–547.

[84] R. LIU, X. LI, L. DU, S. ZHI, AND M. WEI, *Parallel implementation of density peaks clustering algorithm based on spark*, Procedia Computer Science, 107 (2017), pp. 442–447.

[85] Y. LIU, X. DU, AND S. MA, *Innovative study on clustering center and distance measurement of k-means algorithm: mapreduce efficient parallel algorithm based on user data of jd mall*, Electronic Commerce Research, (2021), pp. 1–31.

[86] S. LLOYD, *Least squares quantization in pcm*, IEEE transactions on information theory, 28 (1982), pp. 129–137.

[87] W. LU, *Improved k-means clustering algorithm for big data mining under hadoop parallel framework*, Journal of Grid Computing, 18 (2020), pp. 239–250.

[88] X. LU, H. LU, J. YUAN, AND X. WANG, *An improved k-means distributed clustering algorithm based on spark parallel computing framework*, in Journal of Physics: Conference Series, vol. 1616, IOP Publishing, 2020, p. 012065.

[89] G. LUO, X. LUO, T. F. GOOCH, L. TIAN, AND K. QIN, *A parallel dbscan algorithm based on spark*, in IEEE International Conferences on Big Data and Cloud Computing, Social Computing and Networking, Sustainable Computing and Communications, Atlanta, GA, USA, 2016, IEEE, pp. 548–553.

[90] J. MACQUEEN ET AL., *Some methods for classification and analysis of multivariate observations*, in Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol. 1, Oakland, CA, USA, 1967, pp. 281–297.

[91] X. MALLIOS, V. VASSALOS, T. VENETIS, AND A. VLACHOU, *A framework for clustering and classification of big data using spark*, in On the Move to Meaningful Internet Systems: OTM 2016 Conferences, C. Debruyne, H. Panetto, R. Meersman, T. Dillon, e. Kühn, D. O'Sullivan, and C. A. Ardagna, eds., vol. 10033, Springer International Publishing, Cham, 2016, pp. 344–362.

[92] Y. MAO, D. GAN, D. S. MWAKAPESA, Y. A. NANEHKARAN, T. TAO, AND X. HUANG, *A mapreduce-based k-means clustering algorithm*, The Journal of Supercomputing, 78 (2022), pp. 5181–5202.

[93] F. MARRA, G. POGGI, C. SANSONE, AND L. VERDOLIVA, *Correlation clustering for prnu-based blind image source identification*, in 2016 IEEE International Workshop on Information Forensics and Security (WIFS), IEEE, 2016, pp. 1–6.

[94] D. MELO, S. TOLEDO, F. M. AO, R. SACHETTO, G. ANDRADE, R. FERREIRA, S. PARTHASARATHY, AND L. ROCHA, *Hierarchical density-based clustering based on gpu accelerated data indexing strategy*, Procedia Computer Science, 80 (2016), pp. 951–961.

[95] P. MICHAUD, *Condorcet—a man of the avant-garde*, Applied stochastic models and Data Analysis, 3 (1987), pp. 173–189.

[96] D. S. MILOJICIC, V. KALOGERAKI, R. LUKOSE, K. NAGARAJA, J. PRUYNE, B. RICHARD, S. ROLLINS, AND Z. XU, *Peer-to-peer computing*, Tech. Rep. HPL-2002-57, HP Labs, 2002.

[97] H. S. NAGESH, *High performance subspace clustering for massive data sets*, Master's thesis, North-western University, 2145 (1999).

[98] M. NANNI AND D. PEDRESCHI, *Time-focused clustering of trajectories of moving objects*, Journal of Intelligent Information Systems, 27 (2006), pp. 267–289.

[99] J. NICKOLLS, I. BUCK, AND M. GARLAND, *Scalable parallel programming*, in 2008 IEEE Hot Chips 20 Symposium (HCS), IEEE, 2008, pp. 40–53.

[100] J. OWENS, M. HOUSTON, D. LUEBKE, S. GREEN, J. STONE, AND J. PHILLIPS, *Gpu computing*, Proceedings of the IEEE, 96 (2008), pp. 879–899.

[101] M. A. PATWARY, D. PALSETIA, A. AGRAWAL, W.-K. LIAO, F. MANNE, AND A. CHOUDHARY, *A new scalable parallel dbscan algorithm using the disjoint-set data structure*, in The International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE Computer Society Press, 2012, p. 62.

[102] ——, *Scalable parallel optics data clustering using graph algorithmic techniques*, in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, Denver, Colorado, 2013, ACM, pp. 49:1–49:12.

[103] M. M. A. PATWARY, S. BYNA, N. R. SATISH, N. SUNDARAM, Z. LUKIĆ, V. ROYTERSHTEYN, M. J. ANDERSON, Y. YAO, P. DUBEY, ET AL., *Bd-cats: big data clustering at trillion particle scale*, in SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2015, pp. 1–12.

[104] A. RODRIGUEZ AND A. LAIO, *Clustering by fast search and find of density peaks*, Science, 344 (2014), pp. 1492–1496.

[105] L. ROKACH, *A survey of clustering algorithms*, in Data mining and knowledge discovery handbook, Springer, 2009, pp. 269–298.

[106] P. C. ROTH, D. C. ARNOLD, AND B. P. MILLER, *Mrnet: A software-based multicast/reduction network for scalable tools*, in SC'03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing, IEEE, 2003, pp. 21–21.

[107] I. K. SAVVAS AND D. TSELIOS, *Parallelizing dbscan algorithm using mpi*, in IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Paris, France, 2016, IEEE, pp. 77–82.

[108] N. SCICLUNA AND C.-S. BOUGANIS, *Arc 2014: A multidimensional fpga-based parallel dbscan architecture*, ACM Trans. Reconfigurable Technol. Syst., 9 (2015), pp. 2:1–2:15.

[109] G. SHEIKHOLESLAMI, S. CHATTERJEE, AND A. ZHANG, *Wavecluster: a wavelet-based clustering approach for spatial data in very large databases*, The VLDB Journal The International Journal on Very Large Data Bases, 8 (2000), pp. 289–304.

[110] S. SHI, Q. YUE, AND Q. WANG, *Fpga based accelerator for parallel dbscan algorithm*, Computer Modelling & New Technologies, 18 (2014), pp. 135–142.

[111] R. SIBSON, *Slink: an optimally efficient algorithm for the single-link cluster method*, The computer journal, 16 (1973), pp. 30–34.

[112] D. SINGH AND C. K. REDDY, *A survey on platforms for big data analytics*, Journal Of Big Data, 2 (2014), p. 8.

[113] A. SINHA AND P. K. JANA, *A novel k-means based clustering algorithm for big data*, in 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI), IEEE, 2016, pp. 1875–1879.

[114] D. SKILLICORN, *Strategies for parallel data mining*, IEEE Concurrency, 7 (1999), pp. 26–35.

[115] S. SLAOUI AND Z. DAFIR, *A parameter-free clustering algorithm based k-means*, International Journal of Advanced Computer Science and Applications, 12 (2021).

[116] S. C. SLAOUI, Z. DAFIR, AND Y. LAMARI, *E-transitive: an enhanced version of the transitive heuristic for clustering categorical data*, Procedia Computer Science, 127 (2018), pp. 26–34.

[117] S. C. SLAOUI AND Y. LAMARI, *Clustering of large data based on the relational analysis*, in 2015 Intelligent Systems and Computer Vision (ISCV), IEEE, 2015, pp. 1–7.

[118] H. SONG AND J.-G. LEE, *Rp-dbscan: A superfast parallel dbscan algorithm based on random partitioning*, in Proceedings of the 2018 International Conference on Management of Data, ACM, 2018, pp. 1173–1187.

[119] C. L. SOTIROPOULOU, S. GKAITATZIS, A. ANNOVI, M. BERETTA, P. GIANNETTI, K. KORDAS, P. LUCIANO, S. NIKOLAIDIS, C. PETRIDOU, AND G. VOLPI, *A multi-core fpga-based 2d-clustering implementation for real-time image processing*, IEEE Transactions on Nuclear Science, 61 (2014), pp. 3599–3606.

[120] J. E. STONE, D. GOHARA, AND G. SHI, *Opencl: A parallel programming standard for heterogeneous computing systems*, Computing in science & engineering, 12 (2010), p. 66.

[121] Z. SUN, G. FOX, W. GU, AND Z. LI, *A parallel clustering method combined information bottleneck theory and centroid-based clustering*, The Journal of Supercomputing, 69 (2014), pp. 452–467.

[122] A. TORRENTE AND J. ROMO, *Initializing k-means clustering by bootstrap and data depth*, Journal of Classification, (2020), pp. 1–25.

[123] N. TSAPANOS, A. TEFAS, N. NIKOLAIDIS, AND I. PITAS, *A distributed framework for trimmed kernel k-means clustering*, Pattern recognition, 48 (2015), pp. 2685–2698.

[124] ——, *Efficient mapreduce kernel k-means for big data clustering*, in Proceedings of the 9th Hellenic Conference on Artificial Intelligence, SETN '16, Thessaloniki, Greece, 2016, ACM, pp. 28:1–28:5.

[125] L. G. VALIANT, *A bridging model for parallel computation*, Communications of the ACM, 33 (1990), pp. 103–111.

[126] S. VAN DONGEN, *Graph clustering via a discrete uncoupling process*, SIAM Journal on Matrix Analysis and Applications, 30 (2008), pp. 121–141.

[127] S. VOULGARIS, D. GAVIDIA, AND M. VAN STEEN, *Cyclon: Inexpensive membership management for unstructured p2p overlays*, Journal of Network and Systems Management, 13 (2005), pp. 197–217.

[128] B. WANG, J. YIN, Q. HUA, Z. WU, AND J. CAO, *Parallelizing k-means-based clustering on spark*, in International Conference on Advanced Cloud and Big Data (CBD), Chengdu, China, 2016, IEEE, pp. 31–36.

[129] J. WANG, D. YUAN, AND M. JIANG, *Parallel k-pso based on mapreduce*, in 2012 IEEE 14th International Conference on Communication Technology, Chengdu, China, 2012, IEEE, pp. 1203–1208.

[130] S. WANG, X. LIU, AND L. XIANG, *An improved initialisation method for k-means algorithm optimised by tissue-like p system*, International Journal of Parallel, Emergent and Distributed Systems, 36 (2021), pp. 3–10.

[131] B. WELTON, E. SAMANAS, AND B. P. MILLER, *Mr. scan: Extreme scale density-based clustering using a tree-based network of gpgpu nodes*, in SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE, 2013, pp. 1–11.

[132] F. WINTERSTEIN, S. BAYLISS, AND G. A. CONSTANTINIDES, *Fpga-based k-means clustering using tree-based data structures*, in The 23rd International Conference on Field programmable Logic and Applications, Porto, Portugal, 2013, IEEE, pp. 1–6.

[133] D. XIA, F. NING, AND W. HE, *Research on parallel adaptive canopy-k-means clustering algorithm for big data mining based on cloud platform*, Journal of Grid Computing, 18 (2020), pp. 263–273.

[134] S. XIA, D. PENG, D. MENG, C. ZHANG, G. WANG, E. GIEM, W. WEI, AND Z. CHEN, *A fast adaptive k-means with no bounds*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2020).

[135] W. YAN, U. BRAHMAKSHATRIYA, Y. XUE, M. GILDER, AND B. WISE, *p-pic: Parallel power iteration clustering for big data*, Journal of Parallel and Distributed Computing, 73 (2013), pp. 352–359.

[136] J. YANG AND X. LI, *Mapreduce based method for big data semantic clustering*, in IEEE International Conference on Systems, Man, and Cybernetics, IEEE, 2013, pp. 2814–2819.

[137] A. A. YILDIRIM AND C. ÖZDOĞAN, *Parallel wavecluster: A linear scaling parallel clustering algorithm implementation with application to very large datasets*, Journal of Parallel and Distributed Computing, 71 (2011), pp. 955–962.

[138] M. ZAHARIA, M. CHOWDHURY, M. J. FRANKLIN, S. SHENKER, AND I. STOICA, *Spark: Cluster computing with working sets*, in Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, Berkeley, CA, USA, 2010, USENIX Association.

[139] M. ZAHARIA, M. CHOWDHURY, M. J. FRANKLIN, S. SHENKER, AND I. STOICA, *Spark: Cluster computing with working sets*, in 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10), 2010.

[140] A. ZAYANI, C.-E. BEN N'CIR, AND N. ESSOUSSI, *Parallel clustering method for non-disjoint partitioning of large-scale data based on spark framework*, in IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 2016, IEEE, pp. 1064–1069.

[141] G. ZHANG, C. ZHANG, AND H. ZHANG, *Improved k-means algorithm based on density canopy*, Knowledge-based systems, 145 (2018), pp. 289–297.

[142] H. ZHANG, T. W. CHOW, AND Q. J. WU, *Organizing books and authors by multilayer som*, IEEE transactions on neural networks and learning systems, 27 (2015), pp. 2537–2550.

[143] H. ZHANG, S. WANG, X. XU, T. W. CHOW, AND Q. J. WU, *Tree2vector: learning a vectorial representation for tree-structured data*, IEEE transactions on neural networks and learning systems, (2018), pp. 1–15.

[144] J. ZHANG, G. WU, X. HU, S. LI, AND S. HAO, *A parallel k-means clustering algorithm with mpi*, in The 4th International Symposium on Parallel Architectures, Algorithms and Programming, IEEE, 2011, pp. 60–64.

[145] Y. ZHANG, F. MUELLER, X. CUI, AND T. POTOK, *Large-scale multi-dimensional document clustering on gpu clusters*, in IEEE International Symposium on Parallel Distributed Processing (IPDPS), IEEE, 2010, pp. 1–10.

[146] W. ZHAO, H. MA, AND Q. HE, *Parallel k-means clustering based on mapreduce*, in Cloud Computing, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2009, pp. 674–679.