TESE DE DOUTORADO Nº 280

DISTANCE-BASED CLUSTERING METHODS FOR LARGE DATASETS

Gustavo Rodrigues Lacerda Silva

DATA DA DEFESA: 30/07/2018

Universidade Federal de Minas Gerais

Escola de Engenharia

Programa de Pós-Graduação em Engenharia Elétrica

DISTANCE-BASED CLUSTERING METHODS FOR LARGE DATASETS

Gustavo Rodrigues Lacerda Silva

Tese de Doutorado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do Título de Doutor em Engenharia Elétrica.

Orientador: Prof. Antônio de Pádua Braga

Belo Horizonte - MG

Julho de 2018

Silva, Gustavo Rodrigues Lacerda.
S586d Distance-based clustering methods for large datasets [manuscrito] / Gustavo Rodrigues Lacerda Silva. – 2018.
xiv, 80 f., enc.: il.

Orientador: Antônio de Pádua Braga.

Tese (doutorado) Universidade Federal de Minas Gerais, Escola de Engenharia.

Bibliografia: f. 70-80.

1. Engenharia elétrica - Teses. 2. Algoritmos - Teses. I. Braga, Antônio de Pádua. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

"Distance-based Clustering Methods for Large Datasets"

Gustavo Rodrigues Lacerda Silva

Tese de Doutorado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Doutor em Engenharia Elétrica.

Aprovada em 30 de julho de 2018.

Por:

Prof. Dr. Antônio de Pádua Braga
DELT (UFMG) - Orientador

Prof. Dr. Eduardo Mazoni Andrade Marçal Mendes
DELT (UFMG)

Prof. Dr. Rodney Rezende Saldanha
DEE (UFMG)

Prof. Dr. Luiz Henrique Zárate Galvez
DCC (PUC-MG)

Prof. Dr. Douglas Alexandre Gomes Vieira
ENACOM

Distance-Based Clustering Methods for Large Datasets

Gustavo Rodrigues Lacerda Silva

Graduate Program in Electrical Engineering

Universidade Federal de Minas Gerais

Supervisor: Prof. Antônio de Pádua Braga

Doctoral Dissertation

Electric Engineering

07/2018

Agradecimentos

Jornada longa e bastante desafiadora, que me fez aprender o que é um doutorado (dedicação, humildade e muito trabalho). Agradeço a Deus pela saúde, força e serenidade concedidas para enfrentar essa longa jornada.

Meus sinceros agradecimentos ao meu orientador, Professor Antônio de Pádua Braga, que sempre indicou caminhos nos momentos mais críticos. Obrigado também por me acolher em seu laboratório e pela oportunidade de atuar em diversos projetos de pesquisa. Minha profunda admiração e respeito por você Professor.

Agradeço ao Professor Rodney, pelas conversas sobre os mais diversos assuntos e também ao professor Renato Mesquita por ceder gentilmente a infraestrutura computacional vital para o desenvolvimento deste trabalho.

A ENACOM especialmente ao Douglas por me acolher durante o período do doutorado e oferecer um ambiente de trabalho rico em idéias e inovação. O meu eterno obrigado!

My most sincere thanks to my English teacher Felicity Lee for her knowledge, guidance, patience and willingness to help me in any way. This work wouldn't be nearly as possible without you.

Mãe, sou eternamente grato pelo carinho, amor e, principalmente, pelas nossas conversas durante esse caminho difícil. Pai, pelo amor, apoio, serenidade... sempre um ombro amigo para conversar. Viviane e família, pela alegria e apoio. Torço sempre por vocês!

À minha amada Érica, pelo eterno amor. Que pessoa fantástica!!! Te admiro muito. Obrigado por estar ao meu lado nessa longa jornada,

você fazendo mestrado e eu doutorado, ufa :-). À família da minha amada, o meu muito obrigado!

Agradeço também aos amigos da UFMG e LITC. Um agradecimento especial ao Luiz, Bryan, Camilo, Fred e Adriano Lisboa pelas conversas e ideias ao longo dessa jornada.

Um agradecimento especial ao Rafael Medeiros e ao Paulo Cirino. Sem a sua ajuda de vocês a minha caminhada seria bem mais difícil. A vocês o meu eterno obrigado.

Por fim, agradeço a UNA pela oprotunidade de ministrar aulas. Um agradecimento especial a Ana Paula Ladeira que abriu as portas da instituição para mim e ao amigo Professor Elson Abreu pelas conversas descontraídas na sala dos professores e no retorno para a casa.

Resumo

Este trabalho apresenta uma metodologia direcionada a problemas de agrupamentos com grandes volumes de dados. O objetivo é projetar algoritmos que tenham a capacidade de processar grandes volumes de dados sem a perda de qualidade do agrupamento. Dois novos métodos de agrupamento são propostos. O primeiro é o método de agrupamento *GPIC*, que realiza tanto o cálculo da matriz de afinidades quanto dos autovetores com o auxílio de Unidades de Processamento Gráfico GPUs, do inglês Graphics Processing Unit. O segundo método, denominado *bdrFCM*, reduz o volume de dados utilizando como princípio básico a borda dos agrupamentos resultantes. Resultados encontrados com bases de dados sintéticas e reais demonstram que as abordagens propostas por este trabalho conseguem processar grande quantidade de dados em tempo menor e reduzir o volume de dados, mantendo a qualidade do agrupamento.

Abstract

This PhD dissertation presents a methodology focused on clustering problems with large data volumes. The goal is to design algorithms that can process large volumes of data without loss of clustering quality. Specifically, this Doctoral dissertation presents two novel, fast and scalable distance-based clustering algorithms well suited to analyse large datasets. The first one is the GPIC clustering method, which performs the calculation of the affinity matrix and the eigenvectors with the support of the Graphics Processing Unit - GPU. The second method, called bdrFCM, reduces the volume of data using the border of the Fuzzy *c*-means cluster results as a fundamental principle. Results found with synthetic and real datasets demonstrate that the approaches proposed by this work can process a significant amount of data in less time and reduce the volume of data, whilst maintaining the quality of the clustering result.

Publications

During the development of this Doctoral Dissertation, some papers have been published:

International Journal

SILVA, G.R.L.; MEDEIROS, R. R.; JAIMES, B.R.A.; TAKAHASHI, C.C.; VIEIRA, D.A.G; BRAGA, A.P.. CUDA-Based Parallelization of Power Iteration Clustering for Large Datasets. *Journal IEEE Access*, v. 5, p. 27263-27271, 2017.

International Conference

SILVA, G.R.L.; OLIVEIRA, L. M.; MEDEIROS, R. R.; GOUSSEVSKAIA, O.; BENEVENUTO, F.. Characterizing Internet Radio Stations at Scale. In: IEEE/WIC/ACM International Conference on Web Intelligence (WI), 2017, Leipzig. *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, 2017., 2017.

Invited Paper Under Revision

SILVA, G.R.L.; NETO, P.C.R.; TORRES, L.C.B.; BRAGA, A.P.. A Fuzzy Data Reduction Cluster Method Based on Boundary Information for Large Datasets. *Submitted to Journal of Neural Computing and Applications - Springer*. Invitation for NCAA journal special issue.

Contents

Symbols Lists ix
Acronyms x
List of Figures xii
List of Tables xiv

1 Introduction **1**
1.1 Motivation 1
1.2 Problem Definition 3
1.3 Main Contributions 3
1.4 Doctoral Dissertation Structure 4

2 Literature Review **5**
2.1 Clustering Analysis 5
2.2 Distance-based Clustering 6
2.2.1 Types of Distance-based Clustering Methods 8
2.2.1.1 Partition Methods 8
2.2.1.2 Hierarchical Methods 8
2.2.1.3 Graph Methods 8
2.2.1.4 Fuzzy Methods 9
2.2.2 Criteria for Clustering Quality 9
2.3 Big Data 10
2.3.1 Volume - Data as a Whole 10
2.3.2 Variety - Data in Several Forms 11
2.3.3 Velocity - Data in Motion 11
2.3.4 Value - Data Driven Evidence 11
2.3.5 Veracity - Data Trust 12

2.3.6 Visualisation - Data Understanding 12
2.4 Big Data Clustering 13
2.4.1 Traditional Clustering Methods Applied in BD Problems . 13
2.4.2 Scalability in Clustering Methods 14
2.5 Parallelization of Clustering Methods with GPU 14
2.5.1 CUDA Architecture Overview 15
2.5.2 CUDA Programming Model 17
2.5.3 GPU Data Transfer 19
2.6 Data Reduction Problem Using Clustering Methods 21
2.7 Conclusion 22

3 GPIC **23**
3.1 Introduction 23
3.2 General Proposal 25
3.2.1 Our Approach GPIC - GPU Power Iteration Clustering . . 27
3.3 Experimental Results and Discussion 32
3.3.1 Experiment I - Speedup Comparison With Serial and Parallels Versions 33
3.3.2 Experiment II - Profiling GPIC Algorithm 34
3.3.3 GPU Memory Transfer Analysis 37
3.4 Real Application and Discussion 38
3.4.1 Runtime 39
3.4.2 Cluster Quality Evaluation 40
3.5 Conclusion 43

4 bdrFCM **44**
4.1 Introduction 44
4.2 The Proposed Approach 45
4.2.1 Boundary Estimation Cluster 47
4.2.2 The q_k Analysis for Cluster Boundary Detection 48
4.2.3 The λ_k Analysis for Boundary Detection 48
4.2.4 The q_k and λ_k Analysis 48
4.3 bdrFCM Algorithm - Boundary Data Reduction Fuzzy c-Means . 52
4.4 Experimental Results and Discussion 55

4.4.1 Computation of Initial Cluster Centers 55
4.4.2 bdrFCM × FCM Runtime Comparison 57
4.4.3 bdrFCM × FCM Cluster Quality Evaluation 60
4.5 Conclusions 66

5 Conclusion **67**

References **80**

Symbols List

$\mathbf{A}$	Affinity matrix
E	Set of vertices of edges
d	Input space dimension.
Δ	Distance matrix
c	Number of centers
C	Number of clusters
f	Constant coefficients
G	Graph
m	Distance metric
N	Number of samples of the dataset
p	Number of *threads*
$\mathbf{X}$	Multivariate array of data
V	Set of vertices
$\mathbf{W}$	Row-normalized affinity matrix

Acronyms

ALU	Arithmetic Logic Unit
BD	Big Data
bdrFCM	Boundary Data Reduction Fuzzy c-Means
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies
CACTUS	Clustering Categorical Data Using Summaries
CHAMELEON	Hierarchical Clustering Algorithm Using Dynamic Modeling
CLARANS	Clustering Algorithm based on Randomized Search
CURE	Clustering Using Representatives
CUDA	Compute Unified Device Architecture
DENCLUE	Density Based Clustering
DBSCAN	Density Based Spatial Clustering of Applications with Noise
EM	Expected Maximization
FCM	Fuzzy c-Means
G-DBSACN	Gpu Density Based Clustering
GPIC	Gpu Power Iteration Clustering
GPU	Graphics Processing Unit
OptiGrid	Optimal Grid Clustering
PIC	Power Iteration Clustering
SIMD	Single Instruction Multiple Data
ROCK	Robust Clustering using LinKs
SM	Streaming Multiprocessor
UVA	Unified Virtual Addressing

List of Figures

2.1 GPU CUDA architecture (Kirk & Hwu, 2016). 16
2.2 GPU memory hierarchy (Cheng *et al.*, 2014) 17
2.3 CUDA grid, block and thread structure (Kirk & Hwu, 2016) . . . 18
2.4 CUDA execution flow (Kirk & Hwu, 2016) 19
2.5 Pinned memory (Cheng *et al.*, 2014). 20
2.6 Unified virtual addressing (Cheng *et al.*, 2014) 21

3.1 GPIC split data into chunks and iteratively copied to the device. . 29
3.2 GPIC execution flow. 30
3.3 Affinity matrix representation on GPU. 31
3.4 Kernels launched on GPU and runtime. 34
3.5 NormMat($\mathbf{A}, \mathbf{d}, p$) kernel distilled. 35
3.6 GPIC stall reasons. 36
3.7 GPIC memory access overhead analysis. 38
3.8 GPIC method running in multiple GPUs enviroment. 39
3.9 GPIC segmentation images result for two clusters using Median filter. The entire dataset was provided by the Instituto de Estudos Avançados da Aeronáutica IEAV, São José dos Campos, Brazil. . 42

4.1 Fifty largest (black) and fifty smallest (red) λ_k. 47
4.2 Higher magnitudes near the center of the generator distributions and lower magnitudes near their borders. 49
4.3 Cluster boundary detection with fifty smallest q_k. 50
4.4 The fifty smallest λ_k values for two-norm dataset. 50
4.5 Pearson correlation between q_k and λ_k for two norm dataset. . . . 51

4.6 Pearson correlation between q_k and λ_k for shapes dataset. 51
4.7 The red points show the boundary points detected from bdrFCM for different values of β in the first iteration in two-norm dataset. 52
4.8 Four iterations for bdrFCM Algorithm 3 in two-norm dataset, β=20% and c=2. The red points show the boundary region detected/removed in each iteration step. 53
4.9 Two-norms dataset runtime with different β=[10%, 15%, 20%] values. 58
4.10 Shapes dataset runtime with different β=[10%, 15%, 20%] values. 59
4.11 Runtime results for Poker dataset. 60
4.12 Runtime results for PAMPAP2 dataset. 61
4.13 Runtime results for MNIST dataset. 62
4.14 Runtime results for CoverType dataset. 63
4.15 Runtime results for SKIN dataset. 64

List of Tables

3.1 Runtime (seconds) of PIC algorithm (m=2). 28
3.2 Runtime (in seconds) and speedup comparison of PIC (sequential version of the GPU code running it on just one thread) and GPIC methods on two synthetic datasets. The parameters for all experiments are maxiterations=3, ϵ=0.00001/N, d=2 (dimension), and euclidean distance similarity function. 33
3.3 Runtime in seconds and the percentage gain of GPIC modified version. 36
3.4 Runtime (in seconds) and speedup comparison of PIC (sequential version of the GPU code running it on just one thread) and GPIC modified version. 37
3.5 Three multiple GPUs scenarios to analyze GPIC algorithm performance in AWS cloud environment for aerial images dataset. The parameters for all experiments are maxiterations=3, ϵ=0.00001/N, k=2, and euclidean distance similarity function. 41
3.6 Cluster quality metrics for aerial images dataset. 42

4.1 Datasets summary. 56
4.2 Initialization methods comparison. 57
4.3 Two-norms dataset (c=2) evaluation in external cluster quality metrics with different volume sizes. 61
4.4 Shapes (c=4) dataset evaluation in external cluster quality metrics with different volume sizes. 62
4.5 External cluster quality metrics evaluation of three real world datasets. 63

4.6 p-value results for Jaccard cluster quality metric in Friedman Align Tests. 64
4.7 p-value results for Folkes Mallows metric in Friedman Align Tests. 65
4.8 Align Rank Post for Folkes Mallows metric. 65
4.9 Align Rank Post for Folkes Mallows metric. 66

Chapter 1

Introduction

This chapter presents an overview of this Doctoral dissertation. It comprises motivation, problem definitions and main contributions. The subsequent sections explain each one of these points.

1.1 Motivation

The progressive incorporation of data collection and communication abilities into consumer electronics is gradually transforming our society, affecting labour relations, and creating new social habits. Consumer electronics, as well as industrial and commercial equipments, are continuously incorporating functions that go far beyond their basic purpose (Bi *et al.*, 2014). The synergy between these physical and computational elements form the basis of a profound transformation in global economy. This new scenario expands current infrastructure of data acquisition and processing technology, and significantly enhances connectivity, communication, computation, control technology, and decision making capabilities (Bi *et al.*, 2014).

In this context, many current, real-world problems may involve petabytes of data with thousands of variables that tend to rise continuously if the current growth rate is maintained (Hashem *et al.*, 2015). Exponential growth of data storage capacity in the worldwide network of devices is forecasted for the next few years.

Despite the barriers and the challenges involved, the analysis of large volumes of data has a supreme importance. This new knowledge acquired through techniques, models, theories and tools could allow revolutionary and innovative discoveries in a vast array of areas (engineering, medicine, trade, education, public security, private sectors) (John Walker, 2014).

Exploring, processing and analysing the data from unstructured or semi-structured sources to extract valued knowledge is a difficult task, which has not been entirely solved. The classic methods, algorithms, frameworks and tools for data management have become inadequate for processing large volumes of data (Bello-Orgaz *et al.*, 2016).

Exploring problems with Big Datasets exposes the issues of handling too many files and working with massive data (Leskovec *et al.*, 2014). These types of problems demand rapid movement and operations on that data, alongside being able to work with an enormous diversity of data (Dobre & Xhafa, 2014). Other challenges include access to vast quantities of unstructured data, determination of how much data is enough and the ability to process data flows dynamically (Dobre & Xhafa, 2014).

Though, given the extensive heterogeneous datasets, one of the significant difficulties is to recognize the valuable datasets and how to interpret them to identify useful knowledge-enhancing decision making (Gandomi & Haider, 2015).

BD issues also arise from widespread data sharing, permitting many users to investigate or examine the same dataset. All these require a new action and a new set of complementary techniques. Exploring Big Datasets is the new way to treat and investigate existing and new data sources (Wu *et al.*, 2014).

Being massive, multi-source and heterogeneous, are part of the dynamic characteristics of BD problems (Tsai *et al.*, 2015). To exploit this new resource, it is necessary to scale up and scale out infrastructures and conventional techniques (Dobre & Xhafa, 2014).

Currently, BD processing essentially depends on parallel programming models. Efficient parallel, concurrent algorithms and implementation methods are required to connect the scalability and performance demands caused by BD problems (Dobre & Xhafa, 2014).

1.2 Problem Definition

Most current clustering and data analysis algorithms need to be adapted in the face of this new reality. For instance, new challenges for dataset clustering in this context are: firstly, high dimensionality, due to the vast amount of sources generating and storing data; secondly, large volumes of data, since the amount of data collected can also be huge due to increasing local storage capacity and high sampling throughput.

This has already caused a noticeable impact on present clustering algorithms, implemented in sequential machines since they are usually data intensive and rely on distance metrics, which are calculated in all dimensions. The objective of this PhD dissertation is to study and develop a distance-based clustering techniques well-suited to analyzing large volumes of data.

Thesis. *Although the massive operations are commonly performed for traditional distance-based clustering methods, these operations can be fully understood and adapted for acceleration using parallelism and dedicated hardware. The desired goal is to create techniques capable of reducing the volume of data while maintaining the quality of the clusters results. Thus, diminishing the computational cost associated with a large amount of data.*

1.3 Main Contributions

This Doctoral dissertation presents two novel, fast and scalable clustering algorithms to analyze large datasets:

- The Method GPIC - GPU Power Iteration Clustering is a new clustering algorithm, based on the original PIC proposal, adapted to take advantage of the GPU architecture, maintaining the algorithm's fundamental properties. The proposed method was compared against the serial implementation, achieving a considerable speedup in tests with synthetic and real datasets. A significant volume of real data application ($> 10^7$ records) was used, and we identified that GPIC implementation has good scalability as well as the ability to handle datasets with millions of data points. Our implementation

efforts are directed towards two aspects: to process large datasets in less time and to maintain the same quality of the cluster's results generated by the original PIC version.

- The Method bdrFCM - Boundary Data Reduction Fuzzy c-Means is a new clustering method, based on the original Fuzzy c-Means proposal, adapted to detect and remove the boundary regions of clusters. Our implementation efforts are aimed towards two aspects: processing large datasets in less time and reducing the data volume, maintaining the quality of the clusters. A significant amount of real data application ($> 10^6$ records) was handled, and we identified that bdrFCM implementation has good scalability and therefore the ability to handle datasets with millions of data points.

The algorithms proposed in this work were evaluated with the traditional implementations, achieving a considerable speedup in synthetic and real large datasets. The results proved that the algorithms could be applied in real-world applications with highly accurate results.

1.4 Doctoral Dissertation Structure

This Doctoral dissertation text is organised into five chapters. The current introductory chapter, followed by Chapter 2, which presents a set of basic concepts that are used during the thesis, followed by the description of the principal works related to this Doctoral dissertation. Chapter 3 and 4 discuss the proposed clustering methods GPIC and bdrFCM respectively. Chapter 5 presents the discussions, conclusions, and future works.

Chapter 2

Literature Review

This chapter presents the inside background knowledge related to this thesis. The first section 2.1, presents a clustering analysis overview, followed by an introduction of Distance-based clustering (DBC) methods in section 2.2. Big Data (BD) concept and definitions for this term are introduced in section 2.3. The main challenges for DBC methods with BD problems are presented in section 2.4. Finally, section 2.5 introduces the parallelisation techniques for clustering methods using dedicated hardware, and section 2.6 presents the data reduction techniques found in literature which paves the way for the proposed technique in this PhD work. The last section concludes the chapter.

2.1 Clustering Analysis

Most clustering methods work with a multivariate matrix of data, $\boldsymbol{X}$, which contains the values of the variables that describe each object to be clustered:

$$\boldsymbol{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,j} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ x_{i,1} & x_{i,2} & \cdots & x_{i,j} \end{pmatrix}$$

The input x_{ij} in $\boldsymbol{X}$ represents the value of the j variable in the i object. The matrix of variables $\boldsymbol{X}$ can have the most diverse types of data (absent, repeated, continuous, ordinal).

The data clustering analysis can be defined by the following proposition: Given a multivariate data matrix representation $\boldsymbol{X}$, the goal is to find k groups based on a metric such that the similarities between the objects of the same group are high, while the similarities between objects of different groups are low (Gan *et al.*, 2007).

The authors (Han *et al.*, 2011), (Aggarwal & Reddy, 2013) present some considerations on cluster analysis, such as:

- Partitioning criteria: the choice of an adequate criterion for determining the partition of the set $\boldsymbol{X}$ is an important consideration; that is, stating whether or not a given partition would be satisfactory.

- Grouping Separation: exclusive (belongs to only one group), not exclusive (belongs to more than one group).

- Similarity measures: all clustering methods are based on some measure of similarity or dissimilarity between the data (Gan *et al.*, 2007) . The choice of the similarity function depends highly on the data representation and also on the purpose of the clustering method. In general, the measure of similarity will encapsulate much prior knowledge about them and should be defined or learned in cooperation with the experts of the problem (Everitt *et al.*, 2011) .

Clustering analysis can often be related to desirable characteristics that can be computed from the data. The assignment of determining a clustering method is made difficult by the fact that in many applications more than one of the enumerated features is relevant. The choice of the clustering method strongly depends on the aim of the problem and the context. No technique is optimal in a general sense, and understanding of the distinct characteristics of each method is essential for making appropriate decisions.

2.2 Distance-based Clustering

An important principle for recognising clusters of observations is to detect how close the patterns are to each other, or how far apart. Distance-based clustering

(DBC) methods, can be used with different data types, as long as an appropriate distance metric is created for this data. So, an important definition for DBC methods is the distance metric.

Definition 1. ***A distance metric m*** *is characterised by a function* $m(x_1; x_2)$ *that returns a scalar measure of dissimilarity of how far apart the two objects* x_1 *and* x_2 *are.*

To be considered a metric, the function $m(x_1; x_2)$ must have the following properties (Lin *et al.*, 1998):

- $m(x_1, x_2) \geq 0$ nonnegativity.
- $m(x_1, x_2) = 0$ iff $x_1 = x_2$: reflexivity.
- $m(x_1, x_2) = m(x_2, x_1)$ symmetry.
- $m(x_1, x_2) + m(x_2, x_3) \geq m(x_1, x_3)$ triangular inequality.

The existence of a distance metric m defines a metric space and topologies in this space, which have intrinsic properties that characterise problems of pattern identification. Essential features, such as the structure of the dataset and how they interact with categories in the input space, are directly related to the metric. Thus, each metric characterises a topology, their groups, an arrangement of data and the way they describe them. Once the distance metric m was defined and calculated, the DBC methods need to store these distances in a distance matrix, which is defined below.

Definition 2. ***A distance matrix*** Δ*, defines a pairwise relationship of a dataset* $\boldsymbol{S}$*.*

The distance matrix Δ is formally presented in the equation below:

$$\Delta = \begin{bmatrix} \Delta^s_{1,1} & \Delta^s_{1,2} & \cdots & \Delta^s_{1,C} \\ \Delta^s_{2,1} & \Delta^s_{2,2} & \cdots & \Delta^s_{2,C} \\ \vdots & \vdots & \ddots & \vdots \\ \Delta^s_{C,1} & \Delta^s_{C,2} & \cdots & \Delta^s_{C,C} \end{bmatrix} \tag{2.1}$$

where the $\Delta^s_{i,j}$ is the submatrices of Δ and C is the number of the clusters in the dataset. The submatrices $\Delta^s_{i,i}$ represent, intra-group relationship; otherwise, the submatrices $\Delta^s_{i,j} \forall i \neq j$ represent the inter-groups i and j relationships.

Finally, once the distance metric m and the distance matrix Δ are defined, the DBC methods try to generally optimize a global criteria based on the distance between the objects.

2.2.1 Types of Distance-based Clustering Methods

Distance-based clustering methods can be categorised into a variety of techniques. The main techniques identified in the literature are outlined.

2.2.1.1 Partition Methods

Partition methods divide the feature space into areas of influence based on the distance to the centres of the partitions. Thus, each group can be represented by the centre of its elements. The best known partitioning methods are k-means (MacQueen *et al.*, 1967), Mean Shift (Fukunaga & Hostetler, 1975), k-medoids (Park & Jun, 2009), k-modes (Huang, 1997), and CLARANS (Ng & Han, 2002).

2.2.1.2 Hierarchical Methods

Hierarchical methods organize the data into a hierarchical structure according to the similarities between the data. These techniques do not require the number of groups to be provided by the specialist. The most established hierarchical methods are: BIRCH (Zhang *et al.*, 1996), CURE (Guha *et al.*, 1998), ROCK (Guha *et al.*, 1999), and CHAMELEON (Karypis *et al.*, 1999).

2.2.1.3 Graph Methods

Graph methods represent the data and their proximity through a graph $G(V, E)$, where $G(V, E)$, which $V = \{v_1, ..., v_n\}$ is the set of vertices and E is the set of edges. Each vertex represents an element of the dataset, and the existence of an edge connecting two vertices is done based on the proximity between the two

datasets. The most popular graph-based methods are: CACTUS (Ganti *et al.*, 1999), and Spectral clustering (Ng *et al.*, 2002).

2.2.1.4 Fuzzy Methods

Fuzzy methods allow one instance of the problem to belong to two or more clusters, but with different membership. The most well-known fuzzy clustering methods are: Fuzzy c-means (Bezdek *et al.*, 1984), and Fuzzy k-modes (Huang & Ng, 1999).

2.2.2 Criteria for Clustering Quality

Although the selection of a clustering method depends on the circumstances and the clustering purpose, studies comparing clustering quality methods are valuable because they add to the perception of the properties of the clustering methods.

There are several metrics available in the literature to evaluate the quality of clusters, and their choice depends on the problem in question (Halkidi *et al.*, 2001). The criteria for analyzing the quality of the groups can be based on statistical indexes that quantitatively measure the quality of a partition (Halkidi *et al.*, 2002a,c).

In the external criteria, the quality of the clusters is measured according to a previous division established from the dataset (Rendón *et al.*, 2011). Typically, a C reference partition is used to perform validation. Such validation can be performed through the contingency matrix (Albatineh *et al.*, 2006), which contains the frequency of the objects in the reference clusters C and in the resulting clusters C'.

The internal criteria, however, evaluate the structure of the cluster using only the sample of data, without taking into account the prior information of the group (Rendón *et al.*, 2011).

In the case of the external indexes, the evaluation of the quality of the grouping can vary hugely, because some indexes indicate that the larger the value is , the better the quality of the clustering. For other internal indexes, the opposite is true.

2.3 Big Data

Big Data (BD) can be defined as large volumes of data available in different levels of complexity, generated at different speeds and with different levels of ambiguity, which cannot be processed using traditional techniques (Chen & Zhang, 2014), (Krishnan, 2013).

Some authors use the definition of the three V's (Volume, Variety and Velocity) to present the BD concept (Hashem *et al.*, 2015), (Davis, 2012), (O Reilly Media, 2015), (Zikopoulos *et al.*, 2011) (Dong & Srivastava, 2013), (Reeve, 2013). Other authors go further and add three more V's (Value, Veracity and Visualization) to the BD concept (Hitzler & Janowicz, 2013), (Van Rijmenam, 2014), (Labrinidis & Jagadish, 2012). These definitions will be presented in more detail in the next section.

2.3.1 Volume - Data as a Whole

Volume is represented by a large dataset consisting of data from several sources like the web, sensor and social networks (Sivarajah *et al.*, 2017).

Definitions of BD volumes are relative and vary by factors, such as velocity and the variety of data. What is defined as a BD volume today may not coincide with the same definition in the future because many technologies can be modified (Sivarajah *et al.*, 2017).

Storage capacities will increase, the ability for even bigger datasets to be taken. Further, the type of data discussed under variety, defines what is meant by 'Big'. Two datasets of the same size may require diverse data management technologies based on their type (video, text, etc.). Consequently, these considerations ensure it is impossible to determine a particular threshold for BD volumes (Gandomi & Haider, 2015).

Per day, there are about 2.5 quintiles of bytes produced around the world and by 2020, 43 million gigabytes will have been created (Information, 2015), (O Reilly Media, 2015).

2.3.2 Variety - Data in Several Forms

Variety is represented by the data generated from several sources. The data types include video, image, text, audio, and data logs, in either a structured or an unstructured format (Hashem *et al.*, 2015),(O'Leary, 2013). Data variety can be divided into the items below:

- Structured: formal schema and data models.
- Unstructured: no pre-defined data model.
- Semi-structured: lack of strict data model structure.
- Mixed: various types together.

2.3.3 Velocity - Data in Motion

Velocity defines the need to receive, process and analyse a huge volume of data almost in real-time (Berman, 2013),(Ranjan, 2014). In some applications, the speed of data creation is even more important than the volume. For example, the capacity to process data in real or nearly real-time can achieve a strategic position for a company in the market (McAfee *et al.*, 2012). The speed of arrival and processing of data can been characterized according to the items below (Assunção *et al.*, 2015):

- Batch: at time intervals.
- Near-time: at small time intervals.
- Real-time: continuous input, process and output.
- Streams: data flows.

2.3.4 Value - Data Driven Evidence

Value defines the economic benefits that the data brings to companies, organizations and society as a whole (Chen *et al.*, 2014). This term clarifies the extraction

of knowledge/value from vast amounts of structured and unstructured data without loss (Furtado *et al.*, 2017).

According to (Davenport, 2014), the creation of value in BD has ordinarily been divided into three situations:

- The reduction of uncertainty in the decision-making process, where data is used as knowledge sources for analytical and predictive models.
- The creation of enhancements in processes, products and services based on insights taken from interpreted data.
- The reduction of costs, where data is used to identify fraud or any problems in the process.

2.3.5 Veracity - Data Trust

Veracity defines what conforms to truth or fact and will determine its level of accuracy. Uncertainty can be caused by inconsistencies, model approximations, ambiguities, mistakes, frauds, duplications, or incompleteness (Sänger *et al.*, 2014).

The analysis of the data presents inferences with certain levels of quality, that is, the results obtained through a large volume of data do not have 100% accuracy, they only indicate the probability of such results occurring starting from a certain forecast (Gandomi & Haider, 2015).

2.3.6 Visualisation - Data Understanding

Visualisation of data is about representing essential information and knowledge more intuitively and efficiently through using different visual arrangements such as a graphical layout. Some tools are needed to support the human understanding of the data in order to convert information from massive datasets into insights (Keim *et al.*, 2013).

Working with BD also breaks the nature of visualization. Two trends were born in this new context; interactivity, and real-time updating. Interactive elements help researchers explore their datasets at various levels of complexity and in different configurations. Further, these interactive visualizations immediately consolidate new data added to datasets (Simon, 2014).

2.4 Big Data Clustering

The high complexity and computational cost sometimes makes clustering methods prohibitive for BD problems. Hence, computational efficiency is one of the main features in the BD Clustering context. Some proposals have been put forward using the computing power of a single machine, multiple machines or multiple GPUS.

2.4.1 Traditional Clustering Methods Applied in BD Problems

The new Big Data scenario requires the investigation of new methods for data summarization (Hesabi *et al.*, 2015), subspace representation (Parsons *et al.*, 2004) and clustering (Shirkhorshidi *et al.*, 2014), with lower computational complexity. The implementation of such methods requires scalable computing platforms that are capable of storing and processing massive and high dimensional data (Bekkerman *et al.*, 2011).

A detailed analysis of some clustering methods, which were performed to evaluate the adhesion of each of them when applied to BD problems, is presented in (Fahad *et al.*, 2014). The criteria adopted by the authors were: ability to work with many types of data; dataset size; number of input parameters; ability to work with outliers; runtime; stability (ability to generate the same groupings regardless of the order in which the patterns are presented to the algorithm); ability to work with high-dimensional problems; and finally, cluster shape. Based on that paper, the cluster methods indicated by the author to work with the BD problems were: Fuzzy c-Means (Bezdek *et al.*, 1984), BIRCH (Zhang *et al.*, 1996), DENCLUE (Hinneburg & Keim, 1998), OptiGrid (Hinneburg & Keim, 1999) and EM (Dempster *et al.*, 1977).

The authors also point out some discussion about the evaluation. No algorithm was able to meet all the proposed criteria. EM and Fuzzy c-Means algorithms presented excellent performance with regard to clustering quality, except when submitted to high-dimensional data. None of the algorithms were able to respect the stability criterion.

2.4.2 Scalability in Clustering Methods

As data volume grows, the scalability of a clustering method becomes a mandatory requirement, since a single machine is not able to deal with data in *terabytes* and/or *petabytes*.

There are several ways to split the volume of data to be processed. One approach is to divide the work between multiple cores within the same CPU or GPU or split the work between numerous machines (Kumari *et al.*, 2015).

The *multi-core* approach has its limitations related to processor, memory and disk. A natural alternative is a multi-core approach to multi-core processing, which allows for scalability of processing, memory and disk (Dean & Ghemawat, 2008), (Chen *et al.*, 2011a).

However, the adoption of this practice entails a high communication cost between the machines. This technique allows the data to be distributed into smaller pieces on different machines, and then such machines use their computational power to solve the problem proposed (Shirkhorshidi *et al.*, 2014).

Parallelization of clustering methods is addressed in two ways in the literature: data-level parallelism and task-level parallelism. Data-level parallelism allows each task to perform the same series of calculations on different data. The focus is to distribute the data through different nodes (servers) to be processed in parallel. Furthermore, task parallelism is achieved when the execution of the algorithm can be divided into segments, some of which are independent and therefore executed concurrently (Chen *et al.*, 2011a).

2.5 Parallelization of Clustering Methods with GPU

Recent advances in consumer computer hardware made parallel computing capability widely available to most users. Applications that make effective use of the Graphics Processing Unit (GPU) have reported significant performance gains (Kirk & Hwu, 2016). In the CUDA (Compute Unified Device Architecture) model, GPU is regarded as a co-processor, which is capable of executing a large number of threads in parallel. A single-source program includes host

codes running on CPU (Central Processing Unit) and also kernel codes running on GPU. Computationally-intensive and data-parallel tasks are implemented as kernel codes in such a way that they can be executed in GPUs (Kirk & Hwu, 2016).

An approach that uses k-means on GPUs is presented in (Wu *et al.*, 2009), where the authors applied k-means to a synthetic dataset with one billion data points and two dimensions. With this test setup, the GPU-version took about 26 minutes, and the serial version of the code took six days. The GPU board used was a GeForce GTX 280 with 240 CUDA cores and 1GB of global memory.

Jianqiang et al. (Dong *et al.*, 2013) presented a GPU accelerated BIRCH (Zhang *et al.*, 1996), version that can be up to 154 times faster than the CPU version with good scalability and high accuracy. The hardware configuration was: Intel Core i5-2320 (CPU), 8GB RAM and NVIDIA Tesla K20 (Kepler Architecture GPU) with 2496 CUDA cores and 5GB of global memory.

Andrade et al. (Andrade *et al.*, 2013) presented the algorithm G-DBSCAN, a GPU parallel version of the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) (Ester *et al.*, 1996). Its implementation with GPUs can be over 100 times faster than its sequential version. The experiments were performed on a Intel Xeon 2.40GHz equipped with a Tesla M2050 with 448 CUDA cores and 3GB of global memory. Our approach to GPU implementation is based on the PIC algorithm (Lin & Cohen, 2010), which will be described in the next section.

2.5.1 CUDA Architecture Overview

The architecture of a GPU may vary from generation to generation but is usually composed of Streaming Multiprocessors (SMs), where each Streaming Multiprocessor has numerous Streaming Processors, (SPs) that share arithmetic logic units (ALUs) and instructions an cache (Kirk & Hwu, 2016). Also, each SP can execute thousands of *threads* per application.

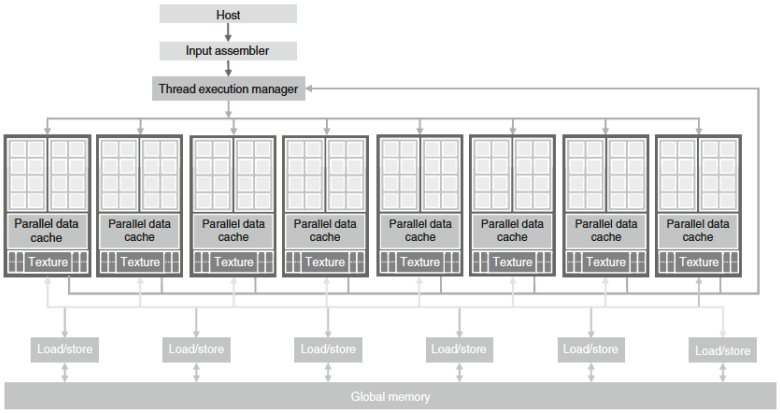


Figure 2.1: GPU CUDA architecture (Kirk & Hwu, 2016).

To maximize performance, the architecture of a CUDA GPU has differentiated memory hierarchies, as illustrated in Figure 2.2. The main hierarchies are (Cheng *et al.*, 2014):

- Register: is the fastest memory space on a GPU.
- Shared memory: basically, for intra-thread communication.
- Global memory: can be statically or dynamically allocated through pointers inside the *kernels*.
- Constant memory: read-only access. Access requests are provided through an optimized *cache* hierarchy to perform the transmission for several threads.
- Local memory: local variables that cannot be manipulated in registers, parameters, and return addresses for subroutines.
- Texture memory: as well as constant memory, read requests of this type are provided by a specific *cache*, optimized for read-only access.

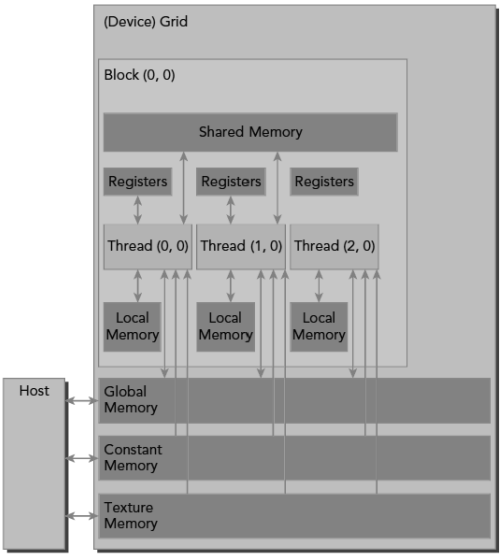


Figure 2.2: GPU memory hierarchy (Cheng *et al.*, 2014)

.

2.5.2 CUDA Programming Model

CUDA is a general purpose parallel architecture designed to utilise the computing power of NVIDIA GPUs. Implementing through a C language extension, allows the developer to control the execution of threads on the GPU and manage its memory (Kirk & Hwu, 2016).

The CUDA programming model considers the GPU as a co-processor capable of executing a large number of threads in parallel. Also, it allows the programmer to develop a code for a single thread (single program multiple data) with the ability to execute in multi-threads and multi-cores (Kirk & Hwu, 2016).

One of the essential concepts in the GPU programming model is the kernel, which can be defined as a piece of code executed in parallel by multiple threads in the GPU. Kernels generate a large number of threads to exploit data-level parallelism. Figure 2.4 illustrates each component of the execution stream of a kernel. Such components are described below (Wilt, 2013):

- Grid: has a matrix-shaped structure, wherein each position of the matrix contains a block. The grid is the highest level of the parallelism hierarchy of a GPU and, is also divided into blocks of threads, limited to the maximum size of 65535 x 65535 x 1 blocks.
- Blocks: each block is allocated to a multiprocessor of the GPU that can execute one or more blocks simultaneously. A block is formed by a pre-defined number of threads (maximum of 512 threads). All blocks created have the same amount of threads.
- Threads: the threads of each block are divided into Warps, each containing 32 threads. GPUs allow simultaneous execution of all threads of Warp, as long as they all execute the same code.
- Warp: is the smallest scaling unit of the GPU and is composed of a group of 32 threads executing in SIMD mode. The occupancy rate of a *kernel* is the ratio of the number of Warps active on the maximum number of Warps supported by the device.

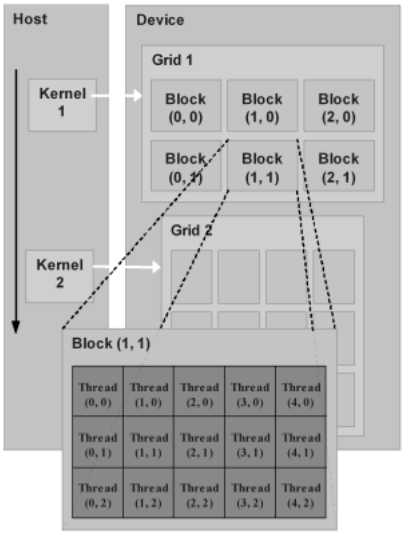


Figure 2.3: CUDA grid, block and thread structure (Kirk & Hwu, 2016)

.

The developer has the responsibility to specify the number of threads per block and the number of blocks of a grid. The number of threads per block also depends on the model of the graphics card (Kirk & Hwu, 2016).

The figure 2.4 shows the execution flow of a CUDA program. The beginning of the program is done by copying the data to be processed in the *host* memory into the main memory of the *device* GPU. Soon after, a grid is configured, and a call is made to a kernel that will run on the GPU. After execution on the device, the programmer must transfer the data from the GPU memory to the CPU and finally free the memory of the device. The execution flow may or may not be terminated, this depends on the implementation performed (Kirk & Hwu, 2016).

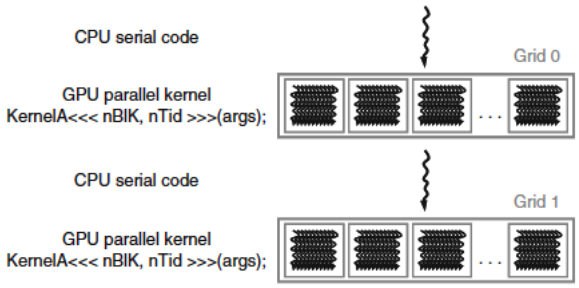


Figure 2.4: CUDA execution flow (Kirk & Hwu, 2016)

.

2.5.3 GPU Data Transfer

The computation of the large datasets in GPUs has some limitations, the main one being the transfer of data between either GPUs and CPUs, or CPUs and GPUs, as these have different sizes and memory speeds. Therefore, understanding the best approach to working with memory allocation in GPUs has extreme importance. The following are the existing approaches (Cheng *et al.*, 2014):

- Pinned Memory: represents a portion of non-paged memory in main memory. The data is allocated directly into the *host* memory area, which allows a direct copy of the *host* non-paged memory into the device memory. Figure 2.5 shows the operation of Pinned Memory.

- Unified Virtual Addressing (UVA): allows *host* memory and *device* memory to be shared in the same memory space, as shown in Figure 2.6.
- Zero-Copy Memory: allows a *kernel* function to be able to access the *host* memory region. Thus, the programmer does not need to copy data from the *host* to the *device*.
- Unified Memory: the main difference from Unified Memory to Unified Virtual Addressing is that the UVA does not migrate the data physically from one device to the other since this capability is unique to Unified Memory. This concept allows you to disengage the memory from the execution so that the data can be migrated on demand to the *host* or *device* to improve locale and execution performance.

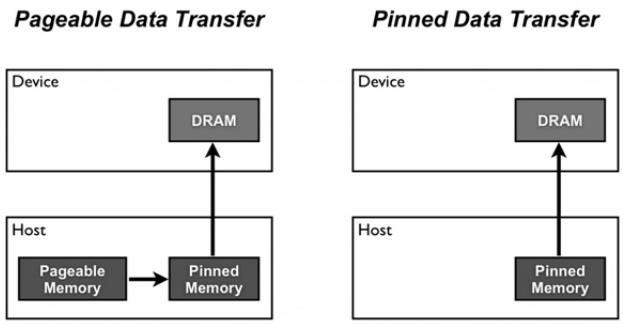


Figure 2.5: Pinned memory (Cheng *et al.*, 2014).

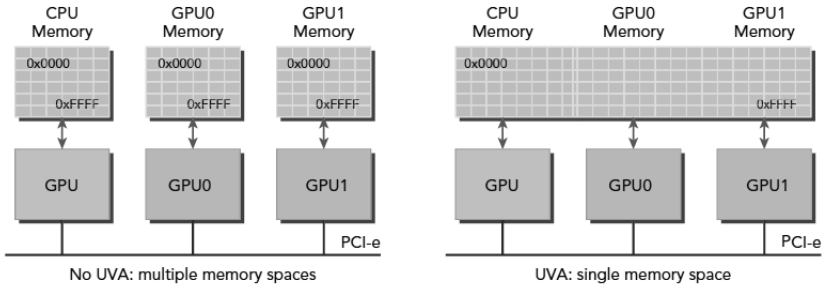


Figure 2.6: Unified virtual addressing (Cheng *et al.*, 2014)

.

2.6 Data Reduction Problem Using Clustering Methods

Data reduction techniques are an essential part of data analysis. It is part of the investigation and a manner of interpretation that sharpens, sorts, focuses, discards and organises data in such a way that the outcomes can be described and validated (Saldaña, 2015). Clustering methods can be used as a support of data reduction techniques. The next few paragraphs will present some strategies outlining how clustering methods are used in the data reduction problem.

A technique that uses the Fuzzy c-Means algorithm in a high volume of data scenario was presented in (Havens *et al.*, 2012). The authors compared three different implementations aimed at extending Fuzzy c-Means in a massive dataset. Firstly, they used a sampling technique; secondly, an incremental method that makes one sequential pass through subsets of the data; and finally, a kernelized version of FCM that provides approximations based on sampling.

Two new accelerated algorithms for the Fuzzy c-Means, that use a statistical method to estimate the subsample size was introduced by (Parker & Hall, 2014). The two proposed approaches were compared with original FCM and four accelerated variants of it. The results found show that the new stopping criterion is effective in speeding up the FCM algorithm without affecting the quality of the cluster's results.

A novel clustering method based on incremental clustering and initial selection for the Big Datasets was presented by Tien et al. (Tien *et al.*, 2017). The proposed algorithm is divided into two steps: firstly, the method determines meshes of rectangles covering data points as the representatives. Secondly, it considers data points that have the high influence on others as the representatives. The representatives are then clustered by FCM, and the new centers are selected as initial ones for clustering of the dataset. The experimental results on synthetic and real datasets show that the total computational time of the new methods, including the time for finding representatives and clustering, is faster than original FCM implementation.

2.7 Conclusion

In this chapter, some of the well-known techniques found in the literature for utilising distance-based clustering methods with large volume dataset problems were analysed. This investigation aimed at identifying the strategies already used to tackle the issue, as well as the significant limitations of the existing state of the art techniques. The next two chapters contain the main part of this Doctoral dissertation; including the two methods developed during this PhD work.

Chapter 3

GPIC

In the previous chapter, we provide a brief problem overview of clustering methods for BD problems. This chapter presents a new clustering algorithm, the GPIC, a Graphics Processing Unit (GPU) accelerated algorithm for Power Iteration Clustering (PIC). The first algorithm designed in this Ph.D thesis. Our algorithm is based on the original PIC proposal, adapted to take advantage of the GPU architecture, maintaining the algorithm's original properties. The proposed method was compared against the serial implementation, achieving a considerable speedup in tests with synthetic and real datasets. A significant volume of real data application ($> 10^7$ records) was used, and we identified that GPIC implementation has good scalability to handle datasets with millions of data points. Our implementation efforts are directed towards two aspects: to process large datasets in less time and to maintain the same quality of the clusters results generated by the original PIC version. The following sections describe our new method in detail.

3.1 Introduction

The implementation of clustering methods for Big Data requires scalable computing platforms that are capable of storing and processing massive and high dimensional data (Bekkerman *et al.*, 2011; Shirkhorshidi *et al.*, 2014). This could drastically limit the usage of classical clustering algorithms like Spectral Cluster-

ing (Von Luxburg, 2007), which is data intensive and requires the calculation of the $n \times n$ distance matrix of the entire dataset and its eigenvalue decomposition.

Although they have proved to be robust, spectral clustering methods have a high order of complexity, especially for calculating the eigenvectors and eigenvalues, which can be as high as $O(n^3)$ (Meila & Shi, 2001). Affinity matrix computation, which represents pairwise relations for spectral clustering, can be a strong limitation when massive data sets are used. Some authors (Chen *et al.*, 2008, 2011b) used parallel implementations to solve the general problem of parallelizing similarity affinity computations. Since pairwise relations are independent, the affinity matrix is particularly appropriate for parallel implementation, which paves the way for adopting end-user parallel machines, such as GPUs (Graphics Processing Units), for massive data implementations (Bekkerman *et al.*, 2011).

In addition, in order to overcome the $O(n^3)$ computation cost of eigenvalue decomposition, parallel implementations have also been presented in the literature (Li *et al.*, 2012). Sparse implementations of the affinity matrix have shown to be a promising strategy to improve scalability (Chen *et al.*, 2011b). Power Iteration Clustering (PIC) (Lin & Cohen, 2010) adopts a different approach, which is based on an approximation of the largest eigenvector, that skips the usual high dimensional matrix operations. The method is based on a normalized pairwise affinity data matrix, which is a valid clustering indicator, consistently outperforming widely used spectral methods.

Our approach, called here GPIC (GPU Power Iteration Clustering), aims at parallelisation of the affinity matrix calculation and the approximation of the largest eigenvector. In order to overcome the computational bottleneck, the GPIC approach aims to create a set of CUDA kernels that can be called one after another to perform different steps of the PIC algorithm in parallel. GPUs offer a massively parallel hardware structure, which is appropriate to overcome some of the difficulties encountered when scaling up clustering methods. In this thesis, experiments were run with 39.936 GPU cores within real dataset problem.

Affinity matrix computation may require regular memory access and abundant parallel computation according to its implementation, which is inherently appropriate for GPU implementation. GPIC's implementation proposes a novel approach to compute the large scale affinity matrix, this implementation splits

the data into chunks, which are iteratively copied to the GPU board device. Using the CUDA framework, a large number of dedicated processors is launched to accomplish the task, which represents a gain in parallelisation when compared to multi-CPU implementation.

GPIC's eigenvector approximation considers the one thread per row approach, where each thread performs a dot product between one row of a matrix and an element of a vector to produce one element of the resulting vector. The main design motivation for this kernel was aimed at data reuse from GPU shared memory, particularly for vector and matrix product computations. Operations were accomplished by block-reading from GPU shared memory, followed by many threads computations on the data. Again, in contrast to other parallel implementations, due to the large number of processors in GPUs, matrix-vector multiplication can be computed in a single parallel step. As presented in the results section, GPIC provides a considerable speedup when compared with the sequential version of the GPU code on a single thread.

3.2 General Proposal

The PIC algorithm is a spectral clustering method, which is aimed at computing the largest eigenvector of a matrix by the Power Iteration Method (Lanczos, 1950). Let's consider $\mathbf{W}$ to be a row-normalized affinity matrix with dominant eigenvalue l_1. So, there is an eigenvector $\mathbf{x}_1$ associated to l_1. Approximation $\hat{\mathbf{x}}_i$ of eigenvector $\mathbf{x}_i$ will be updated at each iteration during PIC execution, hence i is the iteration number and $\mathbf{x}_0$ is the initial vector in the following iteration Eq.(3.1).

$$\hat{\mathbf{x}}_i = \mathbf{W}^i \mathbf{x}_0, \quad i = 1, 2, ..., . \tag{3.1}$$

Let $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$ be the eigenvectors and $l_1, l_2, ..., l_n$ be the corresponding eigenvalues of $\mathbf{W}$. Since $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ are linearly independent, they form a basis of $\mathbb{R}^n$. Consequently, we can write $\mathbf{x}_0$ as a linear combination of these eigenvectors

Eq.(3.2):

$$\begin{aligned}\hat{\mathbf{x}}_i &= \mathbf{W}^i\mathbf{x}_0 \\ &= o_1 l_1^i \mathbf{x}_1 + o_2 l_2^i \mathbf{x}_2 + ... + o_n l_n^i \mathbf{x}_n \\ &= l_1^i \left[o_1\mathbf{x}_1 + o_2 \left(\frac{l_2}{l_1}\right)^i \mathbf{x}_2 + ... + o_n \left(\frac{l_n}{l_1}\right)^i \mathbf{x}_n \right],\end{aligned} \tag{3.2}$$

where $o_1, o_2, ...o_n$ are constant coefficients. Since l_1 is the dominant eigenvalue, it means that each of the ratios $\frac{l_i}{l_1}$ is less than 1 in absolute value Eq.(3.3),

$$\lim_{i\to\infty} \left(\frac{l_j}{l_1}\right)^i = 0, \quad j = 2, ..., n, \tag{3.3}$$

so that Eq.(3.4):

$$\lim_{i\to\infty} \hat{\mathbf{x}}_i = l_1^i o_1 \mathbf{x}_1. \tag{3.4}$$

So, if $l_1 \neq 0$ and $\mathbf{x}_1 \neq 0$, $\hat{\mathbf{x}}_i$ approximates a multiple of $\mathbf{x}_1$ as i increases, that is an eigenvector corresponding to l_1, if $o_1 \neq 0$. In the PIC (Lin & Cohen, 2010) algorithm the Power Iteration Method (Lanczos, 1950) performs the update step Eq.(3.5):

$$\hat{\mathbf{x}}_{i+1} = \frac{\mathbf{W}\hat{\mathbf{x}}_i}{\|\mathbf{W}\hat{\mathbf{x}}_i\|_1}, \tag{3.5}$$

where $\|\mathbf{W}\hat{\mathbf{x}}_i\|_1$ is the L_1 norm of $\mathbf{W}\hat{\mathbf{x}}_i$. A serial version of PIC (Lin & Cohen, 2010) is presented in Algorithm 1.

Algorithm 1 Power Iteration Clustering (Lin & Cohen, 2010)

Input:

$\mathbf{W}$	Row-normalized affinity matrix
kc	Number of clusters
$\mathbf{x}_0$	Initial vector
ϵ	Precision

Output:

$\mathbf{C}$	Clusters $C_1, C_2, ..., C_k$

1: $\delta_0 \leftarrow \mathbf{v}_0$ ▷ Initialize the variation of eigenvector
2: **for** $i = 0, 1, 2, ...$ **do**
3: $\hat{\mathbf{x}}_{i+1} \leftarrow \frac{\mathbf{W}\hat{\mathbf{x}}_i}{\|\mathbf{W}\hat{\mathbf{x}}_i\|_1}$ ▷ Update the eigenvectors of $\mathbf{W}$
4: $\delta_{i+1} \leftarrow |\hat{\mathbf{x}}_{i+1} - \hat{\mathbf{x}}_i|$ ▷ Update the variation of the previous and current eigenvector
5: **if** $|\delta_{i+1} - \delta_i| \leq \epsilon$ **then** ▷ Stop criteria
6: **break** ▷ Terminate the estimation of eigenvectors
7: **end if**
8: **end for**
9: $\mathbf{C} \leftarrow \text{kmeans}(\hat{\mathbf{x}}_i, kc)$ ▷ Cluster eigenvalues
10: **return C**

3.2.1 Our Approach GPIC - GPU Power Iteration Clustering

An experiment was conducted to evaluate the PIC behavior for a moderate amount of data using MATLAB, according to (Lin & Cohen, 2010). We used two synthetic datasets, two moons and three circles with N= 15,000, 30,000 and 45,000 data points. Some notations are defined: N is the number of samples of the dataset, $\mathbf{A}$ is the affinity matrix and d is the input space dimension.

The results of the experiment indicate that the main bottleneck of the PIC algorithm (Lin & Cohen, 2010) is to compute the pairwise distance/similarity for all data points, which is $O(n^2)$ in the worst case. On average, it consumes 88.61% of PIC's total time. The results of the experiment are presented in Table 3.1.

Table 3.1: Runtime (seconds) of PIC algorithm (m=2).

Dataset	N	**A** size[GB]	**A** time [s]	PIC total time [s]	**A** time from PIC total time [%]
2 moons	15.000	3.85	2.534	2.548	99.45%
3 circles			2194.27	2363.64	92.83%
2 moons	30.000	16	18666.39	21680.05	86.09%
3 circles			18064.53	21091.17	85.64%
2 moons	45.000	36.5	97966.50	103778.10	94.39%
3 circles			62228.62	84977.15	73.22%

In order to overcome this bottleneck we propose the GPIC (GPU Power Iteration Clustering) approach, which aims to create a set of CUDA kernels that can be called one after another to perform different steps of the PIC algorithm. According to the growth of the data volume, a larger amount of memory must be available to store the entire affinity matrix, however the GPU memory is limited. Aiming to solve these challenges, our approach splits the data into chunks, which are iteratively copied to the device as presented in Figure 3.1.

The iteratively affinity matrix generation steps are described bellow:

1. Read the input dataset and store in CPU memory.

2. Allocate memory space in the GPU for the input dataset. The chunk size is dynamically set according to the memory size of the GPU board. The input data that is maintained in the CPU memory is copied to the GPU's global memory.

3. Launch a kernel on the GPU board to calculate a chunk of the affinity matrix. It is considered only a portion of the affinity matrix. Then the result is copied back to the CPU memory.

GPIC is described in Algorithm 2 while Figure 3.2 presents all steps of the execution flow. The two main focuses of GPIC are the Affinity matrix calculation and the Power Iteration step.

The first kernel, AffinityMatrix($\mathbf{X}, p$), is launched to evaluate the Affinity matrix $\mathbf{A}$, where $\mathbf{X}$ is the input matrix and p is the number of threads. Matrix $\mathbf{A}$ is symmetrical with dimensions $n \times n$, where n is the number of samples. When

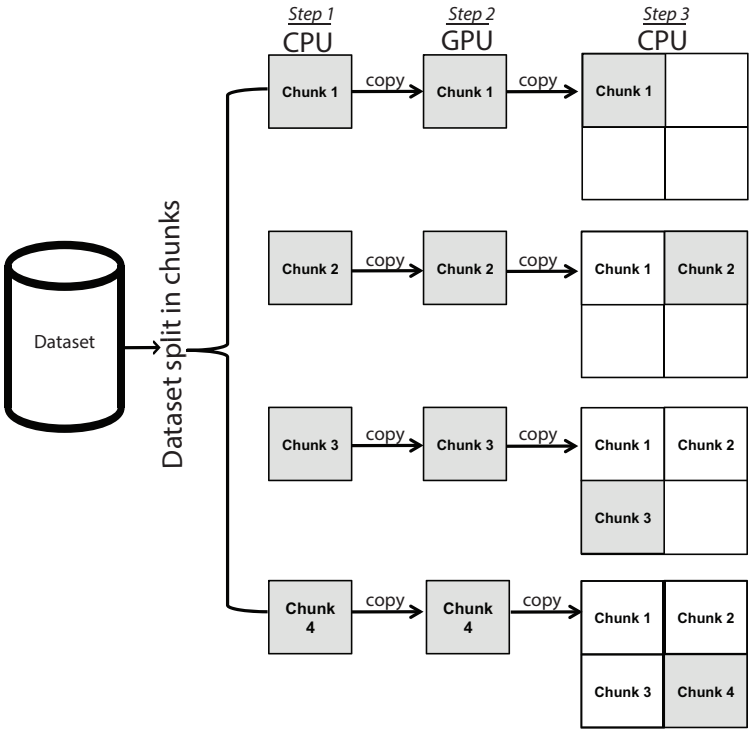


Figure 3.1: GPIC split data into chunks and iteratively copied to the device.

launched, this kernel has p threads and each thread is responsible for calculating a set of $\left\lceil \frac{n}{p} \right\rceil$ rows of matrix $\mathbf{A}$. Thus, each element ij of matrix $\mathbf{A}$ is indexed according to the identifier j of each thread and the current index i. This step has an order of complexity $O(n^2/p)$, and each CUDA core calculates parts of matrix $\mathbf{A}$.

Figure 3.3 presents an example of an affinity matrix $\mathbf{A}$ and this representation on GPU. In this scenario, each thread calculates a row of a matrix $\mathbf{A}$, and every thread performs a control flow structure for each of the elements in every row.

The second kernel RowSum$(\mathbf{A}, p)$ is launched to sum the rows of matrix $\mathbf{A}$ and the result of this operation is stored in vector $\mathbf{s}$. This step has an order of complexity $O(n^2/p)$.

The third kernel, NormMatrix$(\mathbf{A}, \mathbf{s}, p)$, is launched to normalize the affinity matrix $\mathbf{A}$, with the result stored in matrix $\mathbf{W}$. This step has an order of complexity $O(n^3/p)$.

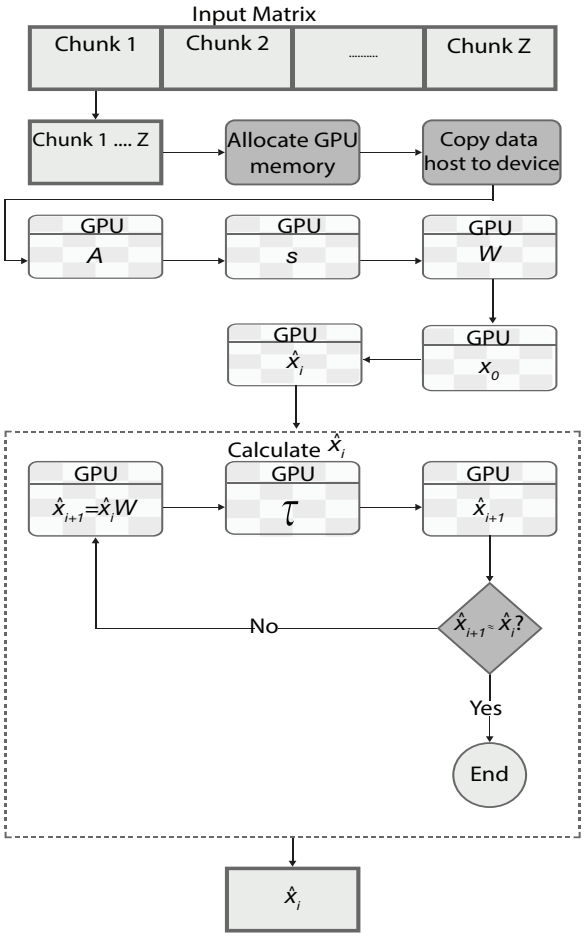


Figure 3.2: GPIC execution flow.

The fourth kernel, Reduction($\mathbf{s}, p$), with an order of complexity $O(n/p + log(p))$, is launched to calculate the sum of vector $\mathbf{s}$. In step 9 of the GPIC Algorithm 2, this operation is called again and the result is stored in τ. Sequential reduction occurs in $O(n)$ and in a thread block where $p < n$ there is $O(n/p + log(p))$ complexity. This pattern is suggested by Nvidia (Harris, 2015) since it is conflict free.

The fifth kernel Norm($\hat{\mathbf{x}}_i$, τ, p) is launched to normalize vector $\mathbf{s}$, with the result stored in $\hat{\mathbf{x}}_i$. The step 10 of the GPIC Algorithm 2 invokes the Norm($\hat{\mathbf{x}}_i, \tau, p$) again in order to update values of $\hat{\mathbf{x}}_i$. The order of complexity of this kernel is $O(n/p)$. The last kernel, Multiply($\hat{\mathbf{x}}_i, \mathbf{W}$), with order of complexity $O(n^2/p)$ is

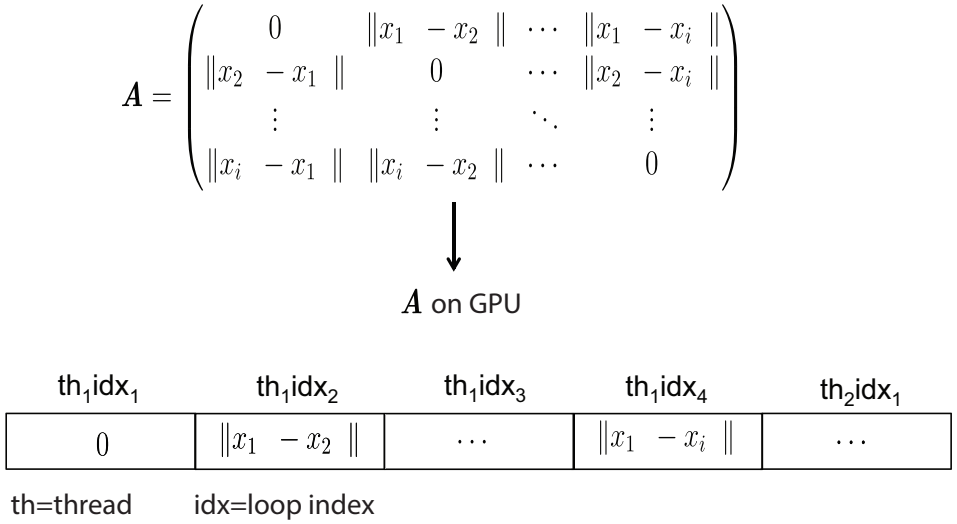


Figure 3.3: Affinity matrix representation on GPU.

launched to multiply matrix $\mathbf{W}$ by vector $\hat{\mathbf{x}}_i$.

Algorithm 2 GPU Power Iteration Clustering

Input:
- $\mathbf{X}$ Input matrix
- $\mathbf{A}$ Affinity matrix
- kc Numbers of clusters
- $\mathbf{x}_0$ Initial vector
- p Number of threads
- ϵ Precision

Output:
- $\mathbf{C}$ Clusters $C_1, C_2, ..., C_k$

1: $\mathbf{A} \leftarrow \text{AffinityMatrix}(\mathbf{X}, p)$ ▷ Affinity matrix kernel
2: $\mathbf{s} \leftarrow \text{RowSum}(\mathbf{A}, p)$ ▷ Sum lines kernel
3: $\mathbf{W} \leftarrow \text{NormMat}(\mathbf{A}, \mathbf{s}, p)$ ▷ Normalize kernel
4: $\mathbf{x}_0 \leftarrow \text{Reduction}(\mathbf{s}, p)$ ▷ Sum kernel
5: $\hat{\mathbf{x}}_i \leftarrow \text{Norm}(\mathbf{s}, \mathbf{x}_0, p)$ ▷ Normalize kernel
6: $\delta_0 \leftarrow \mathbf{x}_0$ ▷ Initialize the variation of eigenvector
7: **for** $i = 0, 1, 2, ...$ **do**
8: $\hat{\mathbf{x}}_{i+1} \leftarrow \text{Multiply}(\hat{\mathbf{x}}_i, \mathbf{W}, p)$ ▷ Multiply kernel
9: $\tau \leftarrow \text{Reduction}(\hat{\mathbf{x}}_{i+1}, p)$ ▷ Sum kernel
10: $\hat{\mathbf{x}}_{i+1} \leftarrow \text{Norm}(\hat{\mathbf{x}}_{i+1}, \tau, p)$ ▷ Update the eigenvectors kernel
11: $\delta_{i+1} \leftarrow |\hat{\mathbf{x}}_{i+1} - \hat{\mathbf{x}}_i|$ ▷ Update the variation of the previous and current eigenvector
12: **if** $|\delta_{i+1} - \delta_i| \leq \epsilon$ **then** ▷ Stop criteria
13: **break** ▷ Terminate the estimation of eigenvectors
14: **end if**
15: **end for**
16: $\mathbf{C} \leftarrow \text{kmeans}(\hat{\mathbf{x}}_i, kc)$ ▷ Cluster eigenvalues
17: **return C**

3.3 Experimental Results and Discussion

The experiments performed in this work were divided into two steps. The first experiment shows the runtime and speedup of the two versions of the PIC Algorithm (sequential version of the GPU code, running it on just one thread and GPIC). The second one evaluated a profiling experiment in the GPIC Algorithm which resulted in a new version of them.

3.3.1 Experiment I - Speedup Comparison With Serial and Parallels Versions

In this section the results of the proposed GPIC algorithm are compared with the sequential version of the GPU code, running it on just one thread version of PIC. Experiments were conducted on a server containing two Intel Xeon E5-2620 (2 GHz, totalling 24 cores) with 64 GB RAM and one k40m model NVIDIA card with 12 GB GDDR5 SDRAM and 2880 CUDA cores running at 745 MHz. Thereby, performance was evaluated using two synthetic datasets (two moons and three circles) with the following matrix dimensions: 15.000 × 15.000 (3.85GB), 30.000×30.000 (16GB) and 45.000×45.000 (36GB). The results presented are the average of ten trials in all experiments.

Table 3.2 shows running times and speedup of PIC and GPIC. Results show a significant speedup of the GPIC Algorithm which, on average, was 188.17 times faster than the original PIC (sequential version of the GPU code running it on just one thread).

Table 3.2: Runtime (in seconds) and speedup comparison of PIC (sequential version of the GPU code running it on just one thread) and GPIC methods on two synthetic datasets. The parameters for all experiments are maxiterations=3, ϵ=0.00001/N, d=2 (dimension), and euclidean distance similarity function.

Dataset	N	A size [GB]	PIC Sequential time [s]	GPIC time [s]	GPIC x PIC Sequential Speedup
2 moons	15.000	3.85	838	3.96	211.61
3 circles			837	4.01	208.72
2 moons	30.000	16	3336	17.57	189.86
3 circles			3336	17.75	187.94
2 moons	45.000	36.5	7499	44.97	166.75
3 circles			7500	45.69	164.14

3.3.2 Experiment II - Profiling GPIC Algorithm

In order to analyze GPIC algorithm performance, a *profile* process was performed with the NVIDIA Visual Profiler tool (NVIDIA, 2016). This tool provides a graphical interface where you can get metrics about the code that was run on the GPU. These metrics present the runtime of each kernel, the amount of global and shared memory used, and the processors usage. This tool also includes an automated analysis engine to identify code optimization opportunities.

The NVIDIA Visual Profiler tool needs a graphical user interface to present results of profile process in a Graphical User Interface (GUI). However, the server where the k40m GPU card is installed does not support a GUI, thus it was necessary to change the simulations to another machine, and consequently another GPU card. This new machine has an Intel Xeon 2.4 GHz, 96 GB of RAM and a graphics card NVIDIA Quadro 4000 with 2 GB of GDDR5 memory and 256 CUDA cores.

Figure 3.4 presents all kernels launched in the GPU and the duration of each of them. The computational cost of each kernel was evaluated, and, although the

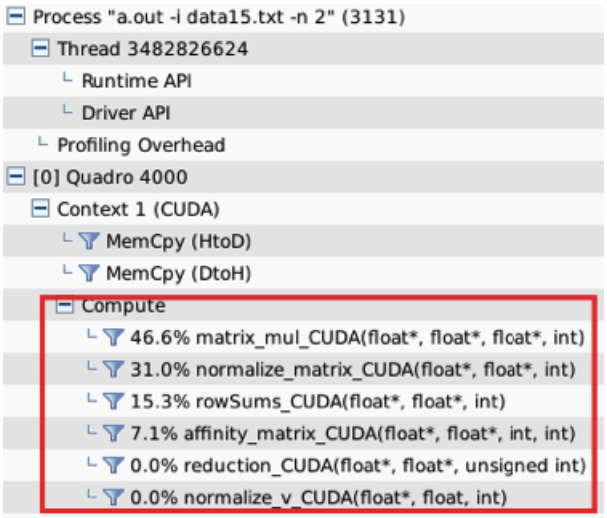


Figure 3.4: Kernels launched on GPU and runtime.

result of the profile shows the Multiply($\hat{\mathbf{x}}_i, \mathbf{W}, p$) kernel with 46.6% of the total cost, in reality this is not the most expensive. This kernel actually runs three

times because of the iterative nature of the GPIC algorithm. The cost of running this kernel when evaluated individually is only 15.53% of the total time. The results indicate that the most expensive kernel was the NormMat($\mathbf{A}, \mathbf{s}, p$) since it consumes 31% of the total time. This occurs because this kernel runs the affinity matrix by performing reading, updating, and writing operations on each of the array elements.

Figure 3.5 shows that there was not an efficient use of global and shared memory on the GPIC code. As a result, a small change in the GPIC code was made. The NormMat($\mathbf{A}, \mathbf{s}, p$) kernel was rewritten to use shared memory instead of global memory. After this modification in the code, another experiment was executed and the corresponding mean execution times are presented in Table 3.3.

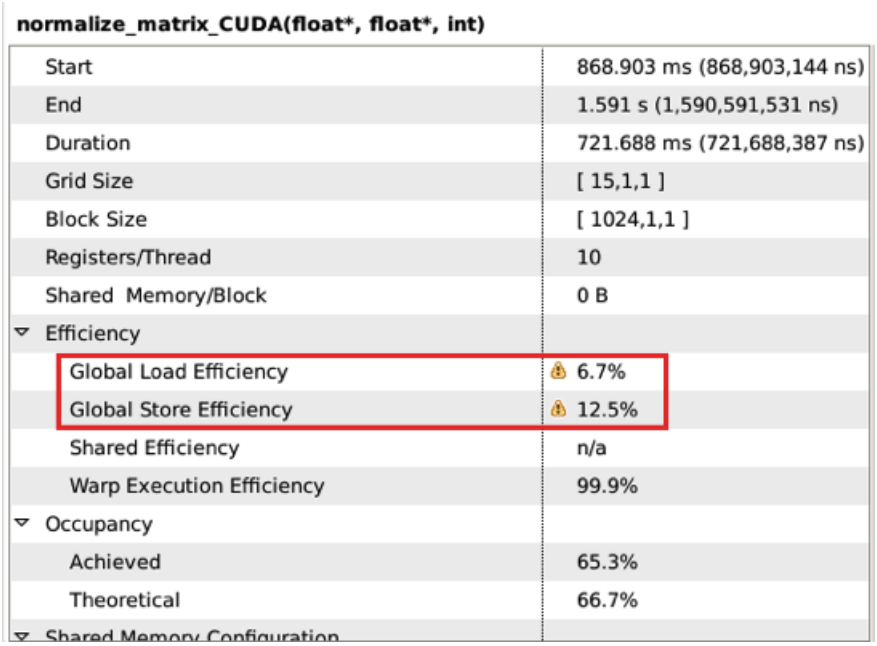


Figure 3.5: NormMat($\mathbf{A}, \mathbf{d}, p$) kernel distilled.

An analysis of the stall reasons on the GPIC code are presented in Figure 3.6. A stall condition occurs when a core cannot run the next instruction because of a dependency on a previous operation. As might be expected, the main reason for processor idleness was memory dependency (47.77%), which corresponds to the

Table 3.3: Runtime in seconds and the percentage gain of GPIC modified version.

Dataset	N	GPIC time [s]	GPIC modified time [s]	Gain GPIC modified [%]
2 moons	15.000	3.96	0.76	19.19 %
3 circles		4.01	0.77	19.20 %
2 moons	30.000	17.57	5.57	31.70 %
3 circles		17.75	5.51	31.04 %
2 moons	45.000	44.97	20.84	46.34 %
3 circles		45.69	20.77	45.45 %

time when the GPU is waiting for the completion of a data transfer between the GPU and the CPU. The second reason for idleness was the dependency between instructions (30.99%). This behavior was also expected due to the iterative nature of the GPIC algorithm. The profiling experiment on the GPIC algorithm was important to identify bottlenecks and their causes. Table 3.3 presents the performance gain on a modified version of the GPIC code over the first version, where the average runtime decreased 32.15%. Finally, Table 3.4 presents the new runtime and speedup of the modified GPIC compared with PIC (sequential version of the GPU code running it on just one thread). Results show a significant speedup of the GPIC Algorithm which, on average, was 685.82 times faster than the original PIC (sequential version of the GPU code running it on just one thread).

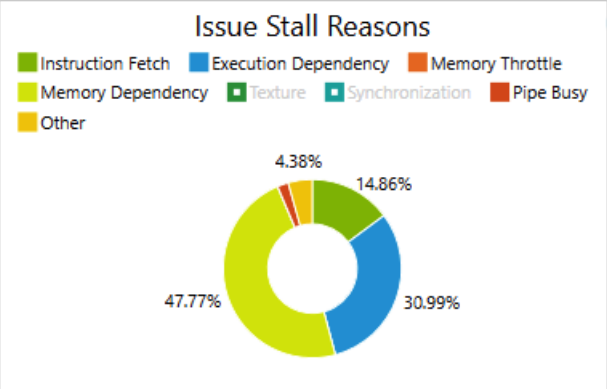


Figure 3.6: GPIC stall reasons.

Table 3.4: Runtime (in seconds) and speedup comparison of PIC (sequential version of the GPU code running it on just one thread) and GPIC modified version.

Dataset	N	A size [GB]	PIC Sequential time [s]	GPIC time [s]	GPIC x PIC Sequential Speedup
2 moons	15.000	3.85	838	0.76	1102.63
3 circles			837	0.77	1087.01
2 moons	30.000	16	3336	5.57	598.92
3 circles			3336	5.51	605.44
2 moons	45.000	36.5	7499	20.84	359.83
3 circles			7500	20.77	361.09

3.3.3 GPU Memory Transfer Analysis

The execution of a kernel in the GPU requires all data used by this kernel to be copied from the CPU to the GPU memory before processing begins. After executing the kernel, the data produced needs to be moved back to CPU memory. Usually, the function used in this type of procedure is a CUDA memory copy operation (*cudaMemcpy*).

An experiment was conducted to analyse the GPU communication/memory access overhead specifically the *cudaMemcpy* operation. A profile process was performed again in the GPIC algorithm focused on this metric. This experiment considers the two moons dataset with 23 different sizes of data volume (5,000, 10,000, 15,000, ... 115,000).

Figure 3.7 presents the GPIC memory access analysis. The percentage of total execution time used in a CUDA memory copy operation is highlighted with a black line and the percentage of the total execution time of all the other CUDA operations combined is presented in gray. Due to the increase of data volume in the GPU board, the operation of data transfer between GPU and CPU presents itself as the step that consumes most of the time.

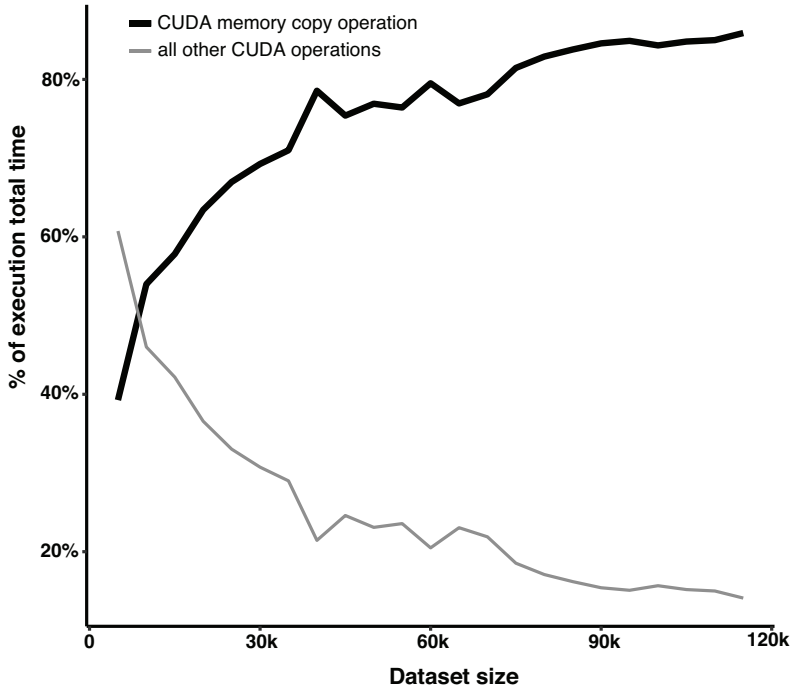


Figure 3.7: GPIC memory access overhead analysis.

3.4 Real Application and Discussion

Many current real-world problems may involve large datasets with thousands of variables that tend to rise continuously if the current growth rate is maintained. In order to test GPIC in a real dataset, an experiment with GPIC was conducted with a very large data set, which turned out to be unfeasible with the original version of serial PIC algorithm.

In this section, results are presented for an image segmentation problem. Experiments were conducted in an Amazon cloud environment that contains a particular GPU instance `p2.16xlarge`. This instance provides general-purpose GPUs along with high CPU performance, large memory and high network speed for applications.

The operational system is Linux, and the hardware configuration is composed of 64 CPU cores, 720 GB RAM, 16 GPUs cards (NVIDIA K80 model) with 12 GB GDDR5 SDRAM with 2496 CUDA cores each board. A new version of the GPIC code was developed to use multiples GPUs on the same server.

Image segmentation relates to partitioning a digital image into multiple regions, based on some criteria. Formally, it is defined as the process of partitioning an image into non-intersected regions, where the pixels of these areas share similar properties. In the experiments performed using the GPIC method, only the colour information was explored during the image segmentation process.

3.4.1 Runtime

To evaluate the performance of the GPIC method in the image segmentation application, 157 aerial images were selected with dimensions of 350×290 resulting in a 101,500 pixels vector. The whole dataset contained 15,935,500 pixels for the 157 images. These images were captured by a drone during a real flight over the city of São Carlos, Brazil. The entire dataset was provided by the Instituto de Estudos Avançados da Aeronáutica IEAV, São José dos Campos, Brazil.

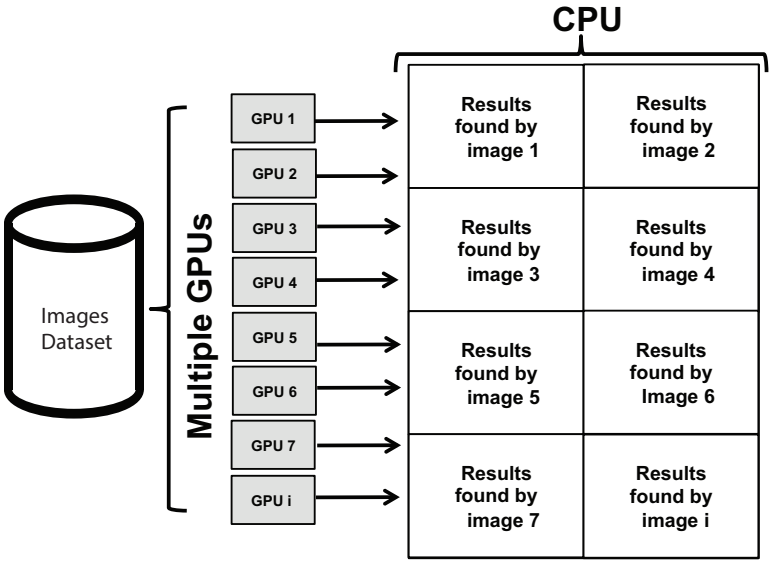


Figure 3.8: GPIC method running in multiple GPUs enviroment.

In order to better understand how multiple GPUs implementation of GPIC can be useful to process the whole dataset (15,935,500 pixels), an experiment was conducted with three multiple GPUs boards configurations:

- Scenario I - 8 GPUs cards (NVIDIA K80 model) and 8 images running at the same time, one on each GPU board;
- Scenario II - 12 GPUs cards (NVIDIA K80 model) and 12 images running at the same time, one on each GPU board;
- Scenario III - 16 GPUs: cards (NVIDIA K80 model) and 16 images running at the same time, one on each GPU board.

In this experiment, each image was copied directly to one of the available GPUs and the GPIC method was immediately applied. After processing in the GPUs, the result was returned to the CPU memory. Since the multiple GPUs have the same computational resources, the images were evenly distributed. Figure 3.8 presents a schematic diagram of the GPIC running in multiple GPUs in the Amazon cloud environment.

The results of this experiment are presented in Table 3.5. As we can see in Scenario I, the average time to process each image in each GPU board is 26.19% faster than Scenario II and 52.59% than Scenario III. This situation can be explained due to the GPU bandwidth saturation of the bus, since the data volume used in Scenario II and III are much bigger than the Scenario I.

To complement the analysis, the total time spent to process the whole dataset was evaluated. Scenarios II and III have lower total run-times than Scenario I. It can be seen in Table 3.5 that the total time spent in Scenario I is 18.86% bigger than Scenario II and 31.19% bigger than Scenario III. It can be concluded that although the runtime of each GPU is lower in the scenario with a small number of GPU boards, the scenarios that used a larger number of GPU boards presented an impressive reduction in the total runtime, because it has the capacity to process more images at the same time.

3.4.2 Cluster Quality Evaluation

In order to compare cluster quality outcomes, the images were pre-processed with two different filters, Gaussian (Ito & Xiong, 2000) and Median (Lim, 1990). These filters were applied to reduce the details and texture, smoothing the pictures and under certain conditions, they preserve edges. This was required because the

Table 3.5: Three multiple GPUs scenarios to analyze GPIC algorithm performance in AWS cloud environment for aerial images dataset. The parameters for all experiments are maxiterations=3, ϵ=0.00001/N, k=2, and euclidean distance similarity function.

Scenarios	Time in seconds spent for each GPU board			Average total time for each scenario [s]
	Min [s]	Max [s]	Mean [s]	
8 Gpus	161.70	223.30	200.40	3931.84
12 Gpus	187.80	285.70	252.90	3307.93
16 Gpus	249.50	357.90	305.80	2996.84

aerial images have texture and changes in luminosity that must be reduced to obtain better results when the labeling of the pictures is done.

The criteria adopted for assessing the quality of the yielded clusters partition was based on a cluster quality index (Halkidi *et al.*, 2002b). Two types of tests are commonly used to assess the quality of clusters: external and internal Rendón *et al.* (2011). External indexes were chosen to validate the experiments once it had a reference partition to compare the results. In the external criteria, the quality index is calculated according to a reference partition obtained from the dataset. Typically, a reference partition P is used to perform the validation. External indexes are computed in the interval [0,1], and the closer the result is to 1, the better the cluster quality is.

The chosen cluster quality indexes were RI (Rand Index) (Hubert & Arabie, 1985), Jaccard (Jaccard, 1908) and Fowlkes-Mallows (Meilă, 2003). The above-mentioned experiments were performed on the dataset 30 times for statistical significance and the results for all trials are presented in Table 3.6. In all experiments the significance level of the statistical tests used was 5%. To analyze the cluster quality results we applied two statistical tests. The first test applied was a Shapiro-Wilk normality test (Royston, 1992), and the result found was that for all two filters and all metrics (RI, Jaccard and Fowlkes-Mallows) the data results do not follow a normal distribution.

The second test applied was Wilcoxon Mann-Whitney (DePuy *et al.*, 2005) to check if there is a significant difference between the scenarios and to determine

which was the best scenario. It was detected that difference between scenarios exists. The best result found was obtained by the Median filter and the best external cluster quality metric is Fowlkes-Mallows 0.8438.

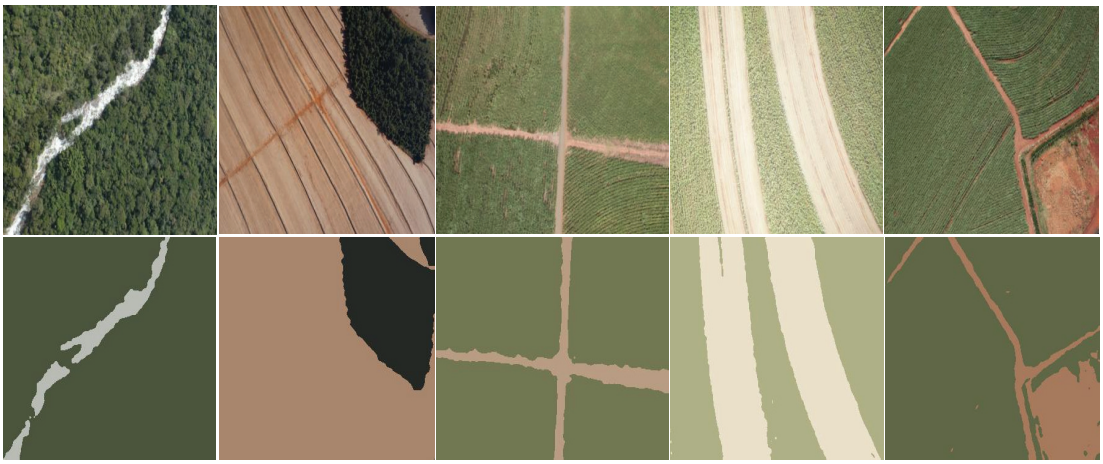


Figure 3.9: GPIC segmentation images result for two clusters using Median filter. The entire dataset was provided by the Instituto de Estudos Avançados da Aeronáutica IEAV, São José dos Campos, Brazil.

According to Table 3.6, it can be concluded that for the two filters (Gaussian and Median) applied to the aerial images dataset, the GPIC algorithm can generate good cluster results for an image segmentation problem.

Table 3.6: Cluster quality metrics for aerial images dataset.

Filter	External indexes		
	Rand Index	Jaccard	Fowlkes-Mallows
Gaussian	0.7093 ± 0.1271	0.7093 ± 0.1271	0.8388 ± 0.0759
Median	0.7178 ± 0.1277	0.7178 ± 0.1277	0.8438 ± 0.0760

Finally, Figure 3.9 presents the GPIC results in the image segmentation process for five images of the dataset. It can be observed that GPIC was able to segment the images very consistently.

3.5 Conclusion

This chapter presented a new clustering algorithm, the GPIC, a CUDA-based parallelisation of Power Iteration Clustering. Our algorithm is based on the original PIC proposal, adapted to take advantage of the GPU architecture. The proposed method was compared with the sequential implementation, achieving a considerable speedup in synthetic and real large datasets.

To the extent of our knowledge, this is the first contribution in the literature that provides a CUDA-based parallelisation of Power Iteration Clustering, which is able to reduce runtime whilst maintaining the cluster quality of the original PIC algorithm.

It is understood that the speedup occurred for two reasons. Firstly the use of parallelism techniques and secondly the availability of a hardware that is suitable to run parallel applications. Matrix operations, which are computationally costly, have been replaced by simpler, similar operations that did not impact upon the final result of the algorithm.

The NVIDIA Visual Profiler tool helped to find bottlenecks in the first version of GPIC code and also estimated the potential performance benefits of removing those bottlenecks. With a small modified version of GPIC code, the average runtime decreased by 32.15%.

Finally, a real-world application from computer vision was explored. Results pointed out that GPIC implementation has good scalability, without suffering performance degradation with large datasets.

GPIC is the first knowledge discovery algorithm designed in this Ph.D thesis. The next chapter describes the other contribution of this thesis; bdrFCM algorithm.

Chapter 4

bdrFCM

The Fuzzy c-Means algorithm (FCM) is aimed at computing the membership degree of each data point to its corresponding cluster center. In this computation it is necessary to calculate the distance matrix between the cluster center and the data point. The main bottleneck of the FCM algorithm is the computing of the membership matrix for all data points. This work presents a new clustering method, the bdrFCM, (Boundary Data Reduction Fuzzy c-Means). Our algorithm is based on the original Fuzzy c-Means proposal, adapted to detect and remove the boundary regions of clusters. Our implementation efforts are directed towards two aspects: processing large datasets in less time and reducing the data volume, maintaining the quality of the clusters. A significant volume of real data application ($> 10^6$ records) was used, and we identified that bdrFCM implementation has good scalability to handle datasets with millions of data points.

4.1 Introduction

The analysis of big datasets presents itself as a challenge in clustering methods. Commonly, this task involves efforts to reduce data to a manageable size because current clustering methods are not computationally capable of handling large data volumes (Fahad *et al.*, 2014). This context could drastically limit the usage of classical clustering algorithms like Fuzzy c-means (Bezdek *et al.*, 1984), which is data intensive and requires the calculation of the distance matrix for the entire dataset.

Data reduction techniques can be considered a useful strategy to handle the heterogeneity and massiveness of big datasets by reducing the high data volume into a manageable size (Li & Nath, 2014). One way to use data reduction in big datasets is to apply sampling approaches (Kleiner *et al.*, 2014),(Liang *et al.*, 2013). Usually, these methods extract some piece of information from big datasets without resorting to high-performance computing.

This thesis proposes a novel data reduction method that extends the Fuzzy c-Means algorithm (Bezdek *et al.*, 1984), called here bdrFCM (Boundary Data Reduction Fuzzy c - Means). Our approach aims to remove the border of clusters through reducing the data volume and maintaining the quality of the cluster's results. As presented in the results section, this implementation provides a considerable speedup when compared with the original version of Fuzzy c-Means. We give a first step towards the analysis of boundary regions to reduce data volume in the Fuzzy c-Means algorithm.

To obtain a deeper understanding of whether the quality of the original FCM was affected by the bdrFCM implementation, the proposed approach was tested with synthetic and real large datasets. The results proved with a statistical test that the cluster quality was not affected. In a real-world dataset (PAMAP2) bdrFCM achieves an interesting speedup; 19.28 times when compared with the original FCM.

4.2 The Proposed Approach

Fuzzy clustering algorithms allow one instance of the problem to belong to two or more cluster groups, this is called membership function. The proposed method, bdrFCM, is based on the principles presented in the paper Pedrycz & Waletzky (1997). The cost function to be minimized is the one presented in Eq. (4.1). The corresponding constrained optimization problem is given by the minimization of J.

$$\begin{aligned} \min \quad & J = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^2 d_{ik}^2, \\ \text{subject to} \quad & \sum_{i=1}^{c} u_{ik} = 1, \quad k = 1, 2, ..., N, \end{aligned} \tag{4.1}$$

where u_{ik} is the membership value of the k-th example, d_{ik}^2, is a distance function that calculates the distance for each pattern and each cluster, N is the number of samples and c is the number of clusters.

Adopting the Lagrange Multipliers solution, the augmented cost function assumes the form of Eq.(4.2).

$$J = \sum_{i=1}^{c}\sum_{k=1}^{N} u_{ik}^2 d_{ik}^2 - \sum_{k=1}^{N} \lambda_k (\sum_{i=1}^{c} u_{ik} - 1). \tag{4.2}$$

The corresponding partial derivatives are presented in Eq. (4.3) and Eq. (4.4).

$$\frac{\partial J}{\partial \lambda_k} = \sum_{i=1}^{c} u_{ik} - 1 : \frac{\partial J}{\partial \lambda_k} = 0 \implies \sum_{i=1}^{c} u_{ik} = 1. \tag{4.3}$$

Differentiating J in relation to an arbitrary u_{ik}, one obtains:

$$\frac{\partial J}{\partial u_{ik}} = 2u_{ik}d_{ik}^2 - \lambda_k : \frac{\partial J}{\partial u_{ik}} = 0 \implies u_{ik} = \frac{\lambda_k}{2d_{ik}^2}. \tag{4.4}$$

Therefore, Eq. (4.5), represents each Lagrange Multiplier of the dataset and give us an interesting insight; the Lagrange Multipliers will be larger for those patterns that have the larger distances in relation to the prototypes of each cluster.

$$\lambda_k = \frac{2}{\sum_{j=1}^{c} \frac{1}{d_{jk}^2}}, \quad k = 1, 2..., N. \tag{4.5}$$

To validate the discovered insight through Eq. (4.5), an experiment was conducted with a bi-variate Gaussian distribution dataset, where the number of clusters equals two ($c = 2$). Fig. 4.1, presents the fifty largest (black) and fifty smallest (red) λ_k respectively. An interesting observation, which can be drawn from Fig. 4.1 is that the largest λ_k values occur in the border of the clusters and not in the separation margin, and the smallest λ_k values occur near the center of the distribution.

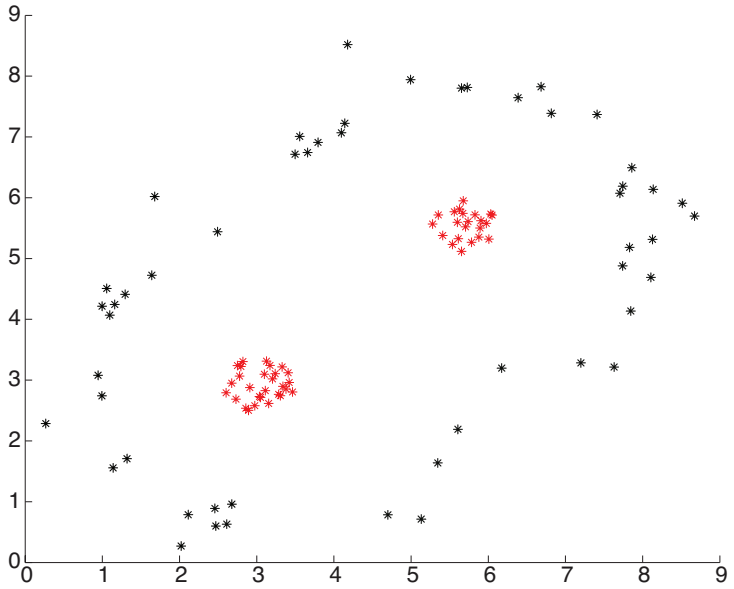


Figure 4.1: Fifty largest (black) and fifty smallest (red) λ_k.

4.2.1 Boundary Estimation Cluster

The Fuzzy c-Means algorithm considers the membership matrix $\mathbf{U} \in R^{c \times N}$, where c is the number of clusters and N is the number of samples. The element $u_{ik} \in \mathbf{U}$ represents the membership of the vector k in relation to cluster i and the elements of the matrix $\mathbf{U}$ are obtained according to Eq. (4.6).

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{d_{ik}}{d_{ij}} \right)^2} \tag{4.6}$$

According to the definition of membership, presented in Eq. (4.7), an equidistant sample from all centers has equivalent membership to $\frac{1}{c}$, as shown in Eq.(4.7) and Eq. (4.8). Samples close to the margin of separation between the clusters tend to have values $u_{ik} \approx \frac{1}{c}$.

$$u_{ik} = \frac{1}{c \left(\frac{d_{ik}}{d_{ik}} \right)^2}. \tag{4.7}$$

$$u_{ik} \approx \frac{1}{c}. \tag{4.8}$$

The analysis of the cluster boundary detection proposed by this article is accomplished by adopting a membership quality measure for each pattern, as described by Eq. (4.9).

$$q_k = 1 - c^c \prod_{i=1}^{c} u_{ik}. \tag{4.9}$$

The quality factor q_k, is a measure of the proximity of a sample to the margin of separation between the clusters. Therefore, samples close to the margin of separation between the clusters tend to minimize Eq. (4.9), while those samples far from the margin of separation tend to be very negative, since small values of u_{ik} result in high values for the term $\prod_{i=1}^{c} u_{ik}$.

4.2.2 The q_k Analysis for Cluster Boundary Detection

In this section we analyze the relationship between the magnitude of q_k for patterns in the cluster group and in the cluster boundaries. The same dataset was used as in the last section and it can be observed in Fig. 4.2 that the q_k values have higher magnitudes near the center of the distributions and lower magnitudes near their borders. In order to observe its relationship with the patterns in the borders, the fifty patterns with smaller q_k are highlighted in black in Fig. 4.3.

4.2.3 The λ_k Analysis for Boundary Detection

In this section we analyze the relationship between the magnitude of λ_k and patterns in the cluster and cluster boundaries. Similarly to what was accomplished for q_k in the last section, Fig. 4.4 presents the fifty smallest patterns of λ_k. It can be seen that their biggest occur in the border of the clusters and not in the separation margin such as in the case of q_k.

4.2.4 The q_k and λ_k Analysis

In this section we conducted an experiment to evaluate the statistical correlation between q_k and λ_k when the number of partitions c varies between the values $\{2, 4, 6, 8, 10, 12, 14, 16, 18\}$. It can be seen in Fig. 4.5 and Fig. 4.6 that there

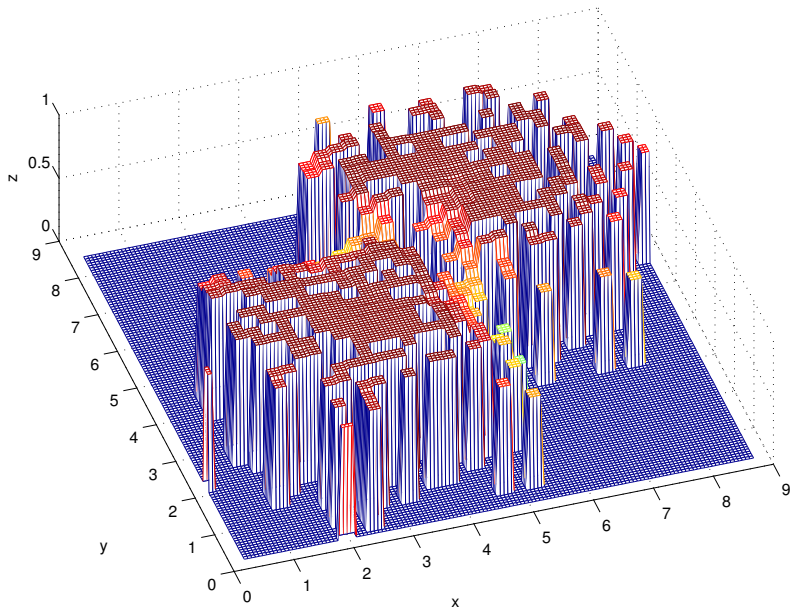


Figure 4.2: Higher magnitudes near the center of the generator distributions and lower magnitudes near their borders.

exists a negative correlation between q_k and λ_k. Another conclusion is that a downward trend exists in negative correlation when the number of partitions c increases.

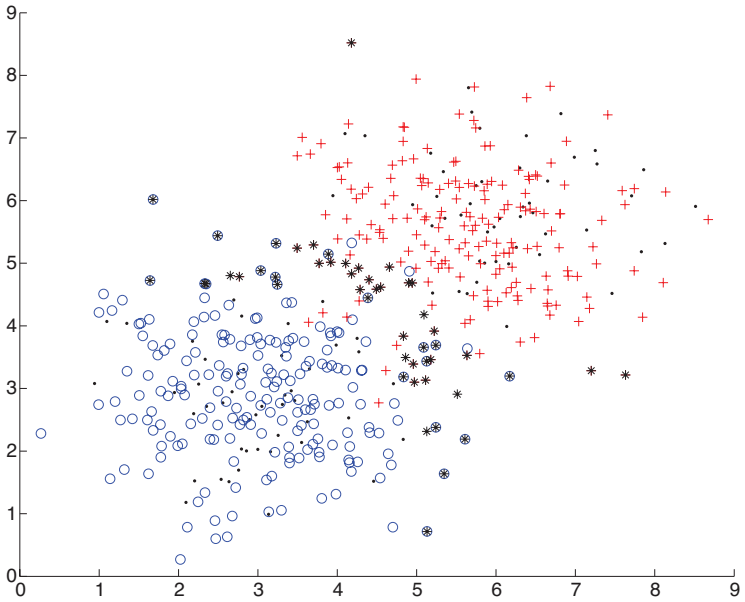


Figure 4.3: Cluster boundary detection with fifty smallest q_k.

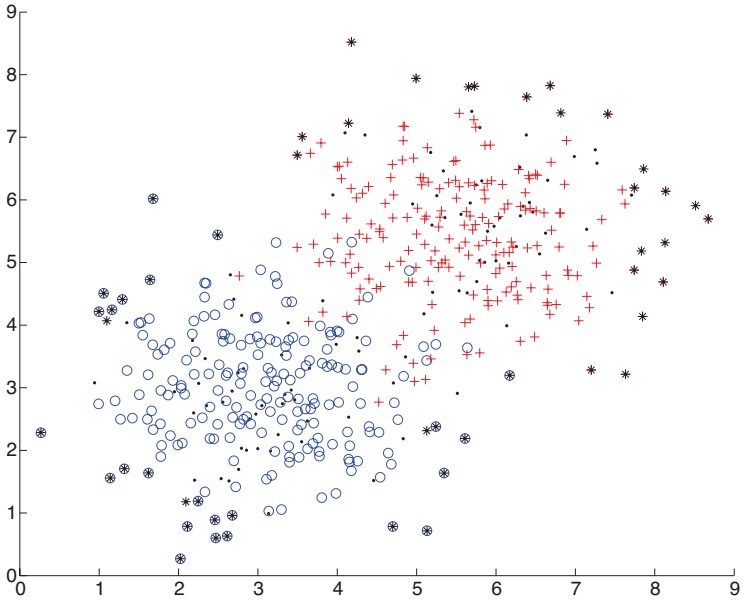


Figure 4.4: The fifty smallest λ_k values for two-norm dataset.

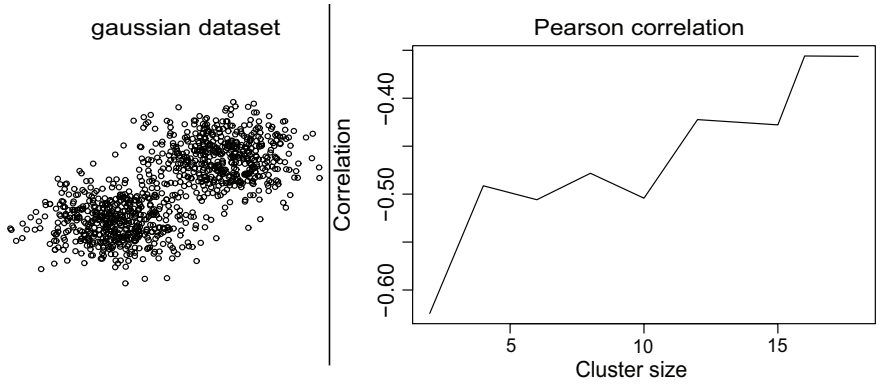


Figure 4.5: Pearson correlation between q_k and λ_k for two norm dataset.

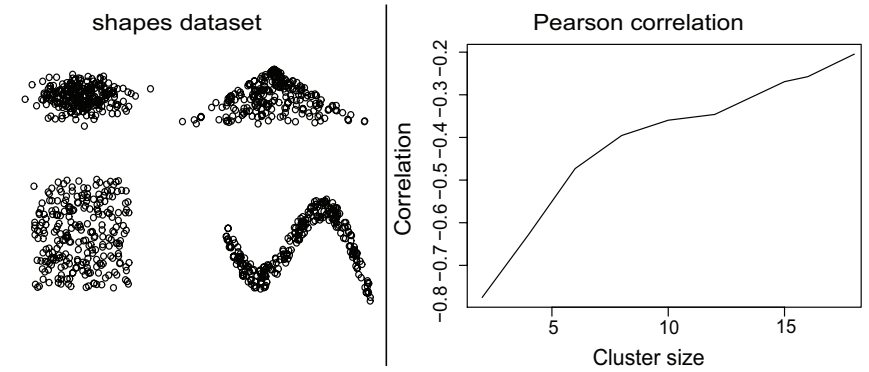


Figure 4.6: Pearson correlation between q_k and λ_k for shapes dataset.

4.3 bdrFCM Algorithm - Boundary Data Reduction Fuzzy c-Means

Reducing runtime is one of the most important factors when working with Big Data problems. This section presents a new clustering algorithm called bdrFCM, based on the original FCM proposal. The proposed method was compared with the original FCM and the results will be presented in the next section. The main objective of bdrFCM Algorithm 3 is to reduce the data volume iteratively and consequently reduce the runtime, maintaining the same quality of the cluster's results generated by the original FCM (Bezdek *et al.*, 1984).

In practice, bdrFCM will often have a shorter runtime than FCM because at each iteration the bdrFCM algorithm removes the boundary points detected by a membership quality measure for each pattern, as described by Eq.(4.9). The size of the boundary region is controlled by a parameter β, this is the only parameter introduced by our approach. Other parameters are the same as in the original version of the Fuzzy c-Means. The bdrFCM algorithm has two stopping criteria; one is to check at each iteration the number of points necessary to represent the boundary region and the other is the original stopping criteria of Fuzzy c-Means. See Algorithm 3 for a detailed description of bdrFCM. Figure 4.7 presents the results when bdrFCM is applied in a synthetic dataset two-norm and Figure 4.8 presents the four iteration steps of bdrFCM.

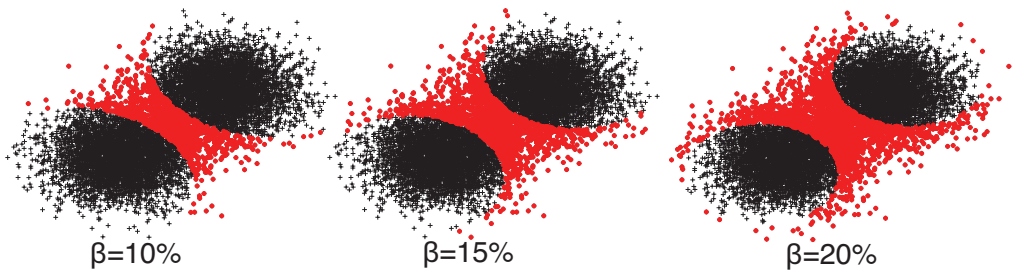
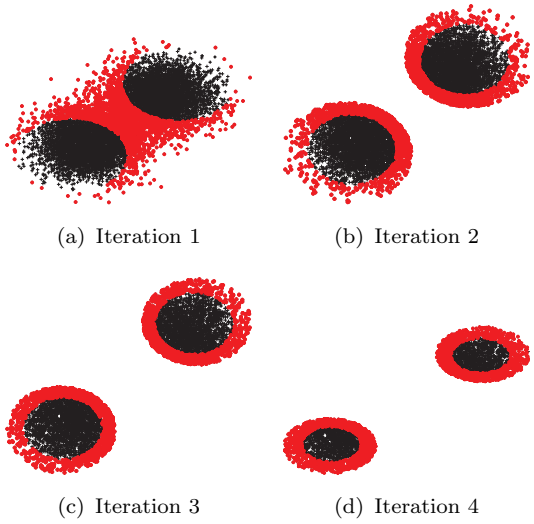


Figure 4.7: The red points show the boundary points detected from bdrFCM for different values of β in the first iteration in two-norm dataset.



(a) Iteration 1 (b) Iteration 2

(c) Iteration 3 (d) Iteration 4

Figure 4.8: Four iterations for bdrFCM Algorithm 3 in two-norm dataset, β=20% and c=2. The red points show the boundary region detected/removed in each iteration step.

Algorithm 3 bdrFCM

Input:

X Dataset.
c Number of centers $2 \leqslant c < n$.
e Weighting exponent $1 \leqslant e < \infty$.
U Fuzzy c partition of **X**.
ε Stop criteria.
β Percentage of points to represent boundary regions.

Output:

C Clusters $C_1, C_2, ..., C_k$

$u_{ik} = \begin{Bmatrix} 1; & x_k \in X_i \\ 0; & otherwise \end{Bmatrix}$

$M_{fc} = \{U_{c \times N} | u_{ik} \in [0,1]\}$ ▷ Randomly initialize the cluster membership values.

while $\left\| \hat{U}^{(k+1)} - \hat{U}^k \right\| < \varepsilon$ **do**

$\hat{u}_{ik} \leftarrow \left(\sum_{j=1}^{c} \left(\frac{\hat{m}_{ik}}{\hat{m}_{jk}} \right)^{\frac{2}{e-1}} \right)^{-1}$ ▷ Compute an updated membership matrix.

$1 \leq k \leq N; 1 \leq i \leq c$

$\hat{m}_{ik} = \|x_k - \hat{c}_i\|$

$\hat{v}_i \leftarrow \frac{\sum_{k=1}^{N} \hat{u}_{ik}^e y_k}{\sum_{k=1}^{N} (\hat{u}_{ik})^e}$ ▷ Compute cluster centers.

$i \leqslant 1 \leq c$

$\hat{U}^{k+1} \leftarrow [\hat{u}_{ik}^{k+1}]$ ▷ Update $\hat{U}^{k+1}$.

$\hat{U}^k \leftarrow \hat{U}^{k+1}$ ▷ Update $\hat{U}^k$.

if $(N \geq N * \epsilon)$ **then**

$\lambda_k \leftarrow \frac{2}{\sum_{j=1}^{c} \frac{1}{m_{jt}^2}}$ ▷ Langrange multipliers, $k = 1, 2..., N$.

$q_k \leftarrow 1 - \frac{2}{\lambda_k} c^c \prod_{i=1}^{c} m_{ik}^2$ ▷ Membership quality measure.

$q_k \leftarrow sort(q_k, \beta)$ ▷ Sort q_k in ascending order.

$BP \leftarrow select(q_k, \beta)$ ▷ Select first βq_k as boundary points BP.

$\mathbf{X} \leftarrow remove(\mathbf{X}, BP)$. ▷ Remove boundary points from **X**.

end if

end while

return C

4.4 Experimental Results and Discussion

The experiments performed in this work were divided into two objectives. The first objective was to evaluate the runtime of the bdrFCM algorithm and the second objective was to evaluate cluster quality. The goal of these experiments was to compare bdrFCM with the original FCM algorithm. The bdrFCM and original Fuzzy c-Means code version used in these experiments was developed in `R` programming language (R Core Team, 2017). The experiment's results are the averages across these 30 trials. The percentage of boundary points β was varied at 10%, 15% and 20% intervals. The scalability efficiency was evaluated by the runtime obtained within different dataset sizes. Experiments were conducted on a server containing two Intel Xeon E5-2620 (2 GHz, totaling 24 cores) with 96 GB RAM, using synthetic and real-world datasets.

4.4.1 Computation of Initial Cluster Centers

In the Fuzzy c-means method, the strategy used for determining initial cluster centres is notably significant in the algorithm performance and has an immediate influence on the structure of final clusters. This implies that the Fuzzy c-means is sensitive to the initial placement of the cluster centers. Proper initialisation of a Fuzzy c-means can avoid some problems such as slower convergence and the chance of getting stuck in local minima.

Fortunately, all of these drawbacks can be treated using different initialisation methods. In this section three methods of choosing initial cluster centres of FCM are evaluated, Random initialization, FCM++ initialization proposed by Adrian et.al (Stetco *et al.*, 2015) and a self-proposed parallel adaptation of FCM++ (Stetco *et al.*, 2015), presented in Algorithm 4. This adaptation (Algorithm 4), runs the initialization steps of FCM++ in all available processors. It finds the best centers solution, which minimizes the sum of distances between points to their centers.

An experiment using five datasets was conducted and the results are described in Table 4.1. The experiments try to check if Algorithm 4 is significantly better than two other methods using a significance level equal to 5%. A Friedman Aligned Ranks Test was applied (Garcia & Herrera, 2008). This test verifies

Algorithm 4 Best centers initialization solution of FCM++ (Stetco *et al.*, 2015).

Input:
X Dataset
k Number of partitions
s Number of initializations in parallel
r Spread factor (r=2)

Output:
R Best centers initialization.

1: $proc \leftarrow s$ ▷ Assign initializations in parallel for each processor available.
2: **foreach** processor proc **do**
3: $R_{proc} \leftarrow R_{proc} \cup$ random point from dataset
4: **while** size of $R_{proc} < k$ **do**
5: sample $x \in X$ with probability $\frac{d^r(x,R_{proc})}{sum(d^r)}$
6: $R_{proc} \leftarrow R_{proc} \cup x$
7: **end while**
8: $Bci \leftarrow mse(R_{proc}, \mathbf{X})$ ▷ Minimize the sum of distances of points to their centers.
9: **end foreach**
10: $BciAll \leftarrow combine(Bci)$ ▷ Combine each R found in each processor.
11: $R \leftarrow R_i \mid i = argmin(BciAll)$
12: **return R** ▷ Best centers initialization solution.

if there is significant difference between blocks with a fast multiple comparison method. Essentially, the procedure involves ranking each row (block) together, then ordering the row's values in decreasing order and calculating the average rank for each column (factor).

Table 4.1: Datasets summary.

Dataset	Groups	Number of samples
Two-Norms (Leisch & Dimitriadou, 2010)	2	5,000
Shapes (Leisch & Dimitriadou, 2010)	4	5,000
Hyper Cubes (Leisch & Dimitriadou, 2010)	4	5,000
Iris (Repository, 2017b)	3	150
Breast Cancer (Repository, 2017a)	4	569

Table 4.2 presents the initialization methods average rank comparison. The smaller values in Table 4.2 represent a better performance of the initialization method. As we can see in Table 4.2, the parallel adaptation proposed by Al-

gorithm 4 received the first rank for all datasets, all of two (internal/external) cluster evaluation metrics proposed in the experiments. Based on the results of Table 4.2 we used the FCM++ Best Initialization as the initialisation method of the bdrFCM algorithm.

Table 4.2: Initialization methods comparison.

Cluster evaluation metric		Initialization method results for Friedman Average Aligned Ranks test applied in datasets described in Table 1		
		Random	FCM++	FCM++ Best Init.
External	Jaccard	2.57	2.43	1
	Folkes Mallows	2.57	2.43	1
Internal	Calinski Harabasz	2.86	2.14	1
	Silhouette	2.57	2.43	1

4.4.2 bdrFCM × FCM Runtime Comparison

To check the scalability of bdrFCM we considered the following synthetic datasets Two-norms, Shapes and Hyper Cubes. The experiments considered the volume data variation 15,000, 45,000, 135,000, 405,000 and
1,215,000.

Figure 4.9 presents the Two Norms dataset runtime results. For 1,215,000 samples, in mean bdrFCM 10% was **1.32** faster, bdrFCM 15% was **1.96** times faster and bdrFCM 20% was **2.52** times faster than the original FCM.

Figure 4.10 presents the Shapes dataset runtime results. For 1,215,000 samples, in mean bdrFCM 10% was **1.23** faster, bdrFCM 15% was **1.82** times faster and bdrFCM 20% was **2.36** times faster than the original FCM. Finally, the good scalability of bdrFCM has also been confirmed in several real world datasets

Poker[1], PAMAP2[2], MNIST[3], CoverType[4] and SKIN[5]. Poker (n = 1,025,010, c=10 *reduced to 3 centers*, d=11): consists of ten groups with 1,025,010 samples. This dataset is extremely unbalanced, there are six groups which together comprise less than 1% of the total number of samples. We merged minor frequent groups together while leaving the large groups untouched. Three final groups were obtained, which correspond to about 50.12% (Group 1), 42.25% (Group 2) and 7.63% (Group 3) of the total number of samples. The next two datasets have a large number of dimensions, so a dimensionality reduction technique PCA (Abdi & Williams, 2010) was applied. Figure 4.11 presents the Poker dataset results; bdrFCM 10% was **3.43** faster, bdrFCM 15% was **5.08** times faster and bdrFCM 20% was **6.62** times faster than the original FCM.

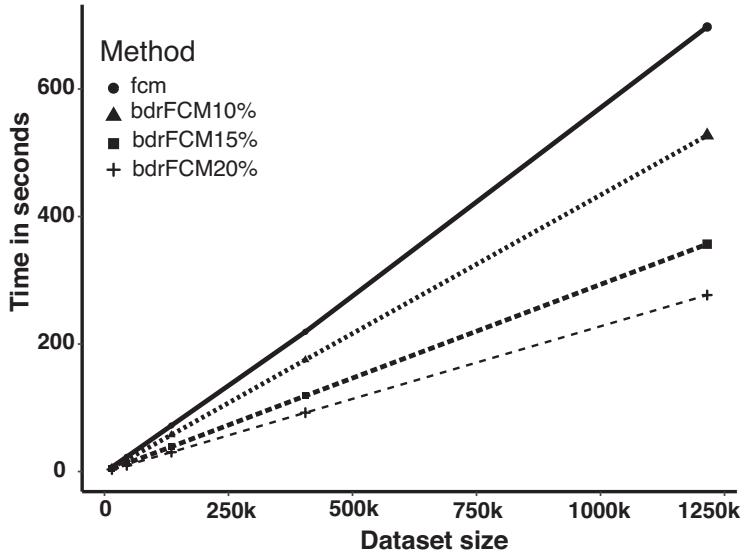


Figure 4.9: Two-norms dataset runtime with different β=[10%, 15%, 20%] values.

The PAMAP2 (n=3,850,505, c=9, d=54 *reduced to 4 dimensions*) is a real-world dataset representing the physical activity monitoring of nine different people. This dataset was preprocessed and reduced from 3,850,505 to 2,844,868. Figure 4.12 presents the PAMAP2 dataset runtime results; bdrFCM 10% was

[1] https://archive.ics.uci.edu/ml/datasets/Poker+Hand (Cattral & Oppacher, 2007)
[2] http://archive.ics.uci.edu/ml/datasets/pamap2+physical+activity+monitoring (Reiss, 2012)
[3] http://yann.lecun.com/exdb/mnist/ (LeCun *et al.*, 1998)
[4] https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/ (Blackard & University, 1998)
[5] https://archive.ics.uci.edu/ml/datasets/skin+segmentation (Rajen Bhatt, 2012)

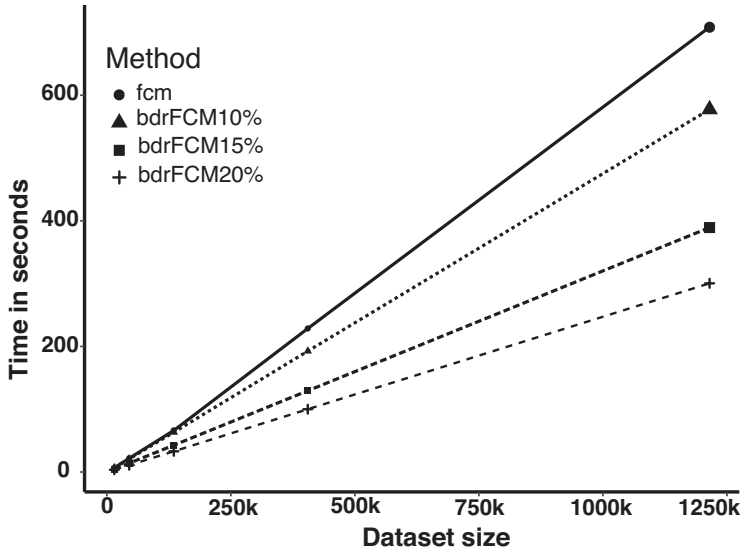


Figure 4.10: Shapes dataset runtime with different β=[10%, 15%, 20%] values.

10.04 faster, bdrFCM 15% was **14.91** times faster and bdrFCM 20% was **19.28** times faster than the original FCM.

The MNIST (n=70,000, c=10, d=784 *reduced to 30 dimensions*) is a collection of handwritten digits. There are 70,000 samples with 28×28 pixel images of the digits 0 to 9. Each pixel has an integer value between 0 and 255. The pixel values were normalised to the interval [0, 1]. Figure 4.13 presents the MNIST dataset runtime results; bdrFCM 10% was **1.53** faster, bdrFCM 15% was **2.27**, times faster and bdrFCM 20% was **2.95** times faster than the original FCM.

The CoverType (n=581,012, c=7, d=54 *reduced to 4 dimensions*) are composed of cartographic variables that are obtained from U.S. government. There were variables such as elevation, wilderness area and soil-type. These features were collected from a total of 581,012 30 × 30 m cells, which were then determined to be one of 7 forest cover groups. The features were normalised to the interval [0, 1]. Figure 4.14 presents the CoverType dataset runtime results; bdrFCM 10% was **2.07** faster, bdrFCM 15% was **3.08**, times faster and bdrFCM 20% was **4** times faster than the original FCM.

The Skin (n=245,057, c=2, d=4) dataset is collected by randomly sampling B, G and R values from face images of various age groups (young, middle, and old) and race groups (white, black, and asian). The dataset is divided as follows:

50,859 samples are labeled as skin samples and 194,198 are labeled as non-skin samples. Figure 4.15 presents the SKIN dataset runtime results; bdrFCM 10% was **1.12** faster, bdrFCM 15% was **1.66**, times faster and bdrFCM 20% was **2.15** times faster than the original FCM.

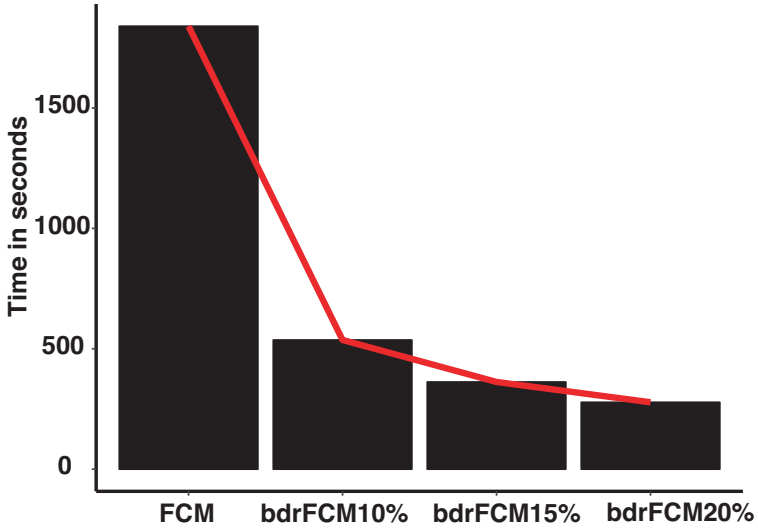


Figure 4.11: Runtime results for Poker dataset.

4.4.3 bdrFCM × FCM Cluster Quality Evaluation

The quality of the resulting partitions was evaluated according to the external cluster validation indexes: Jaccard Index (Jaccard, 1908) and Folkes Mallows (Fowlkes & Mallows, 1983). Tables 4.3, 4.4 and 4.5 present the cluster quality results for all datasets and for the Folkes Mallows and Jaccard metric.

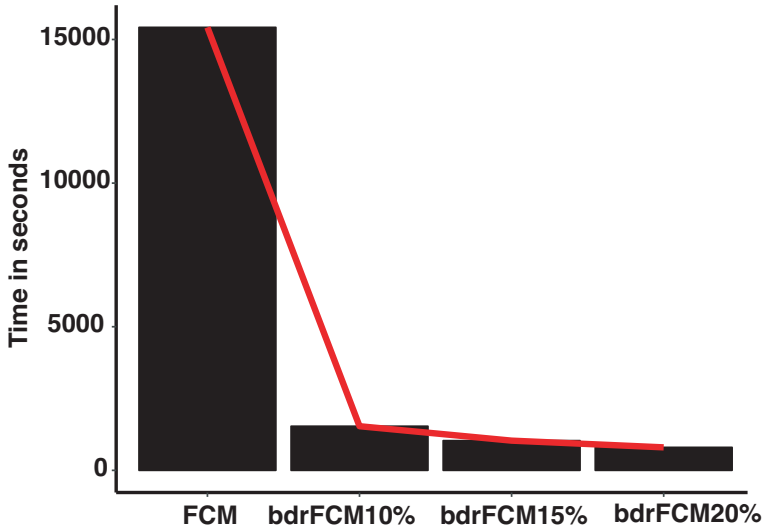


Figure 4.12: Runtime results for PAMPAP2 dataset.

Table 4.3: Two-norms dataset (c=2) evaluation in external cluster quality metrics with different volume sizes.

Volume	Cluster Quality Metric	FCM	bdrFCM 10%	bdrFCM 15%	bdrFCM 20%
15k	Folkes Mallows	0.86	0.86	0.86	0.86
	Jaccard	0.75	0.75	0.75	0.75
45k	Folkes Mallows	0.85	0.85	0.85	0.85
	Jaccard	0.75	0.75	0.75	0.75
135k	Folkes Mallows	0.85	0.85	0.85	0.85
	Jaccard	0.75	0.75	0.75	0.75
405k	Folkes Mallows	0.85	0.85	0.85	0.85
	Jaccard	0.75	0.75	0.75	0.75
1215k	Folkes Mallows	0.85	0.85	0.85	0.85
	Jaccard	0.75	0.75	0.75	0.75

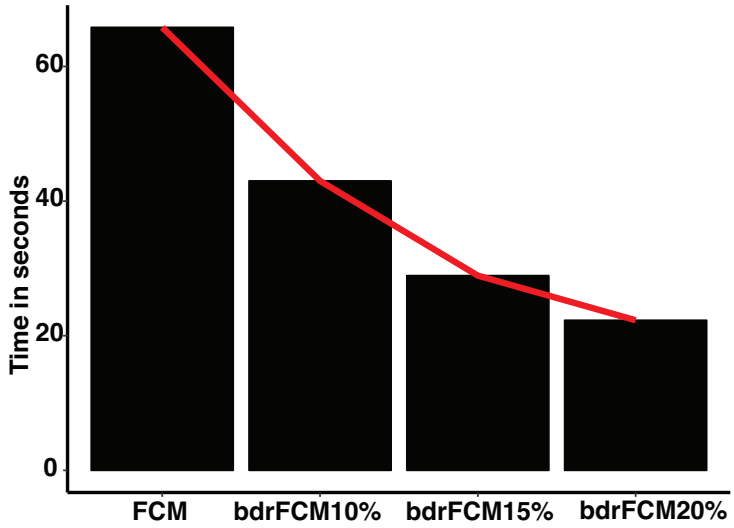


Figure 4.13: Runtime results for MNIST dataset.

Table 4.4: Shapes (c=4) dataset evaluation in external cluster quality metrics with different volume sizes.

Volume	Cluster Quality Metric	FCM	bdrFCM 10%	bdrFCM 15%	bdrFCM 20%
15k	Folkes Mallows	1	0.98	0.99	1
	Jaccard	1	0.95	0.98	1
45k	Folkes Mallows	1	0.99	1	1
	Jaccard	1	0.98	1	1
135k	Folkes Mallows	1	0.98	1	1
	Jaccard	1	0.96	1	1
405k	Folkes Mallows	1	0.98	1	1
	Jaccard	1	0.95	0.99	1
1215k	Folkes Mallows	1	0.98	1	1
	Jaccard	1	0.97	1	1

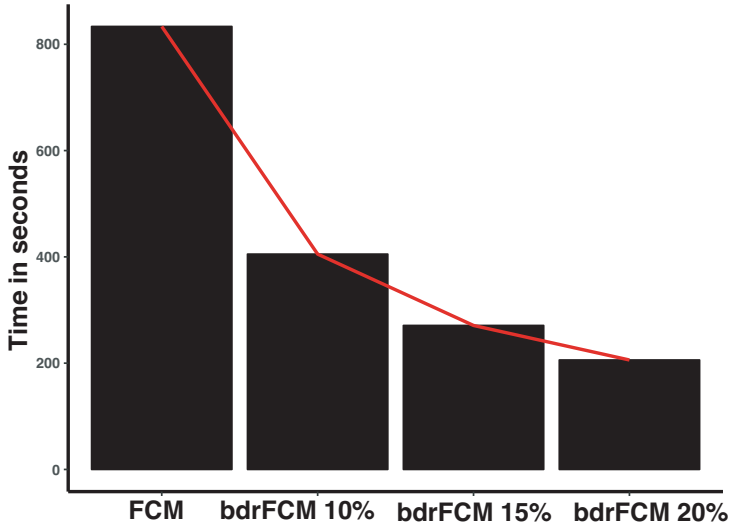


Figure 4.14: Runtime results for CoverType dataset.

Table 4.5: External cluster quality metrics evaluation of three real world datasets.

Dataset	Cluster Quality Metric	FCM	bdrFCM 10%	bdrFCM 15%	bdrFCM 20%
MNIST	Folkes Mallows	0.27	0.24	0.25	0.25
	Jaccard	0.11	0.11	0.13	0.13
Poker	Folkes Mallows	0.38	0.44	0.42	0.40
	Jaccard	0.23	0.28	0.26	0.25
PAMAP2	Folkes Mallows	0.23	0.22	0.23	0.23
	Jaccard	0.13	0.13	0.12	0.13
CoverType	Folkes Mallows	0.43	0.41	0.41	0.40
	Jaccard	0.27	0.25	0.26	0.25
Skin	Folkes Mallows	0.60	0.60	0.60	0.60
	Jaccard	0.43	0.43	0.43	0.43

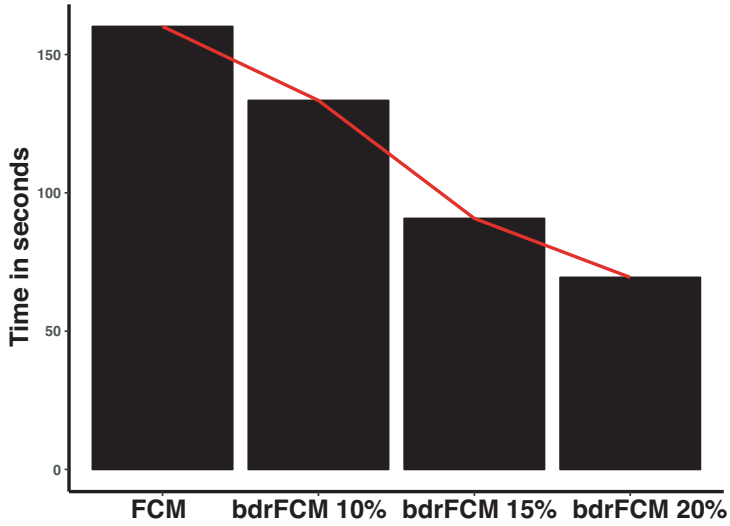


Figure 4.15: Runtime results for SKIN dataset.

Tables 4.6 and 4.7 presents the p-value of the statistical Friedman Test and Table 4.8 and 4.9 presents the ranks used to compare all datasets. It can be concluded that for these datasets there is no evidence that the bdrFCM algorithm has a significant impact on FCM performance. This result is important and shows that the bdrFCM algorithm can generate good cluster results with lower computational cost.

Table 4.6: p-value results for Jaccard cluster quality metric in Friedman Align Tests.

Methods	bdrFCM 10%	bdrFCM 15%	bdrFCM 20%	FCM
bdrFCM 10%	-	0.5929	0.9148	0.9148
bdrFCM 15%	0.5929	-	0.5212	0.5212
bdrFCM 20%	0.9148	0.5212	-	1
FCM	0.9148	0.5212	1	-

Table 4.7: p-value results for Folkes Mallows metric in Friedman Align Tests.

Methods	bdrFCM 10%	bdrFCM 15%	bdrFCM 20%	FCM
bdrFCM 10%	-	0.8306	0.5212	0.4542
bdrFCM 15%	0.8306	-	0.3924	0.5929
bdrFCM 20%	0.5212	0.3924	-	0.1646
FCM	0.4542	0.5929	0.1646	-

Table 4.8: Align Rank Post for Folkes Mallows metric.

Datasets	Methods			
	bdrFCM 10%	bdrFCM 15%	bdrFCM 20%	FCM
Covertype	3	2	4	1
MNIST	4	2	1	3
PAMAP2	3	4	2	1
Poker	1	2	3	4
Skin	3	2	1	4
Avg. Rank	2.8	2.4	2.2	2.6

Table 4.9: Align Rank Post for Folkes Mallows metric.

Datasets	Methods			
	bdrFCM 10%	bdrFCM 15%	bdrFCM 20%	FCM
Covertype	3	2	4	1
MNIST	4	2	3	1
PAMAP2	3	4	2	1
Poker	1	2	3	4
Skin	3	2	1	4
Avg. Rank	2.8	2.4	2.6	2.2

4.5 Conclusions

This chapter presents the bdrFCM, a scalable clustering approach to accelerate the Fuzzy c-means algorithm in the face of large datasets. To the extent of our knowledge, this is the first contribution that provides a boundary detection technique to reduce the data volume in the Fuzzy c-Means algorithm.

It was identified that bdrFCM implementation has good scalability to handle datasets with millions of data points, when compared to traditional FCM. Further to this, an experiment was conducted in order to analyze the impact on cluster quality when the number of samples was reduced. It can be seen that the cluster quality had no significant variations with the bdrFCM algorithm adoption in a synthetic and real world dataset.

Chapter 5

Conclusion

The investigation into clustering methods based on large volumes of data is a field of research with high relevance to all areas of knowledge. This type of study presents significant challenges given the complexity, volume, variety and velocity of the data, among other factors.

In this work the clustering algorithms explored are those within the distance-based category. Its properties were analyzed and exploited with the two main aims; speeding up computing operations and reducing the volume of data. Two distance-based clustering algorithms were chosen to be adapted for BD scenario.

The first one chosen was the Power Iteration Clustering - PIC algorithm. A new version of PIC was created using hardware acceleration (GPU Power Iteration Clustering - GPIC). Graphics Processing Units offers a massively parallel hardware structure, which is appropriate to speed up the original version of PIC. The contribution of the GPIC algorithm is based on the creation of CUDA kernels sets that can be called upon to perform different steps of the PIC algorithm in parallel.

During the analysis of this method two main bottlenecks, as outlined in depth in section 3.2.1, were detected. The first remedy to this was to compute the affinity matrix in a parallel mode and the second is to compute the approximation of the largest eigenvector. Affinity matrix computation may require regular memory access and abundant parallel computation according to its implementation, which is inherently appropriate for GPU implementation. GPIC's eigenvector approximation considers the one thread per row approach, where each thread performs a

dot product between one row of a matrix and an element of a vector to produce one element of the resulting vector.

The GPIC algorithm speed up was proved with a synthetic and a real world dataset problem. Results pointed out that GPIC implementation has good scalability, without suffering performance degradation with large datasets. Another important result is that the proposed method did not impact the quality of the cluster's results.

There are some points that deserve some discussion and further development in future works on the GPIC method. The first is related to the GPU memory transfer operation. As presented previously, the execution of a kernel in the GPU requires all data used by this kernel to be copied from the CPU to the GPU memory before processing begins. After executing the kernel, the data produced needs to be moved back to CPU memory.

This operation can be improved by monitoring bandwidth to buffers, the idea is to avoid re-accessing the same buffer data multiple times if it can be stored in shared memory or cache. An improvement of the method using shared memory was proposed in this work but further analysis can be conducted to fine tune this operation even more. Another relevant topic for future work on the GPIC algorithm is the development of optimization techniques focused on determining the chunk size, the number of GPUs required, and the buffer size on each GPU board.

Another knowledge discovery algorithm designed in this PhD work is the bdrFCM. A new version of fuzzy c-means was proposed to detect and remove boundary regions of clusters. The contribution of bdrFCM is based on a membership quality measure (q_k) for each pattern. The idea is that patterns which are strongly linked to a given cluster group will have the membership q_k close to one and those that have small membership values for all groups will have q_k close to zero. Some equations were developed in Section **??** demonstrate that the metric proposed embodies information about the cluster boundaries.

The bdrFCM will often have a shorter runtime than the original version of fuzzy c-means because at each iteration the bdrFCM removes the boundary points detected by the membership quality measure for each pattern. The parameter β controls the size of the boundary region and it is important to mention that this

parameter is the only one introduced by the bdrFCM proposal. Other settings are the same as in the original version of FCM.

The results found by the bdrFCM algorithm in real and synthetic datasets proved that this algorithm could reduce time without affecting the quality of the cluster results.

Some suggestions of future work in the bdrFCM algorithm could be the development of the smart stopping criteria based on the information of generated clusters and the extension of bdrFCM to parallel implementation.

References

Abdi, H. & Williams, L.J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, **2**, 433–459. 58

Aggarwal, C.C. & Reddy, C.K. (2013). *Data clustering: algorithms and applications*. CRC Press. 6

Albatineh, N.A., Niewiadomska-Bugaj, M. & Mihalko, D. (2006). On similarity indices and correction for chance agreement. *Journal of Classification*, **23**, 301–313. 9

Andrade, G., Ramos, G., Madeira, D., Sachetto, R., Ferreira, R. & Rocha, L. (2013). G-dbscan: A gpu accelerated algorithm for density-based clustering. *Procedia Computer Science*, **18**, 369–378. 15

Assunção, M.D., Calheiros, R.N., Bianchi, S., Netto, M.A. & Buyya, R. (2015). Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, **79**, 3–15. 11

Bekkerman, R., Bilenko, M. & Langford, J. (2011). *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press. 13, 23, 24

Bello-Orgaz, G., Jung, J.J. & Camacho, D. (2016). Social big data: Recent achievements and new challenges. *Information Fusion*, **28**, 45–59. 2

Berman, J.J. (2013). Introduction. In J.J. Berman, ed., *Principles of Big Data*, xix – xxvi, Morgan Kaufmann, Boston. 11

Bezdek, J.C., Ehrlich, R. & Full, W. (1984). Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, **10**, 191–203. 9, 13, 44, 45, 52

Bi, Z., Da Xu, L. & Wang, C. (2014). Internet of things for enterprise systems of modern manufacturing. *Industrial Informatics, IEEE Transactions on*, **10**, 1537–1546. 1

Blackard, J.A. & University, C.S. (1998). Covertype data set. https://archive.ics.uci.edu/ml/datasets/covertype, online; accessed 16-August-2017. 58

Cattral, R. & Oppacher, F. (2007). Poker hand data set. https://archive.ics.uci.edu/ml/datasets/Poker+Hand, online; accessed 16-August-2017. 58

Chen, C.P. & Zhang, C.Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, **275**, 314–347. 10

Chen, M., Mao, S. & Liu, Y. (2014). Big data: A survey. *Mobile Networks and Applications*, **19**, 171–209. 11

Chen, W.Y., Song, Y., Bai, H., Lin, C.J. & Chang, E. (2008). Psc: Paralel spectral clustering. *Software available. http://www. cs. ucsb. edu/˜ wychen/sc*. 24

Chen, W.Y., Song, Y., Bai, H., en Lin, C.J. & Chang, E.Y. (2011a). Large-scale spectral clustering with mapreduce and mp1. *Scaling up Machine Learning: Parallel and Distributed Approaches*. 14

Chen, W.Y., Song, Y., Bai, H., Lin, C.J. & Chang, E.Y. (2011b). Parallel spectral clustering in distributed systems. *IEEE transactions on pattern analysis and machine intelligence*, **33**, 568–586. 24

Cheng, J., Grossman, M. & McKercher, T. (2014). *Professional Cuda C Programming*. John Wiley & Sons. xi, 16, 17, 19, 20, 21

Davenport, T. (2014). *Big data at work: dispelling the myths, uncovering the opportunities*. harvard Business review Press. 12

Davis, K. (2012). *Ethics of Big Data: Balancing risk and innovation*. " O'Reilly Media, Inc.". 10

Dean, J. & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, **51**, 107–113. 14

Dempster, A.P., Laird, N.M. & Rubin, D.B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1–38. 13

DePuy, V., Berger, V.W. & Zhou, Y. (2005). Wilcoxon–mann–whitney test. *Encyclopedia of statistics in behavioral science*. 41

Dobre, C. & Xhafa, F. (2014). Parallel programming paradigms and frameworks in big data era. *International Journal of Parallel Programming*, **42**, 710–738. 2

Dong, J., Wang, F. & Yuan, B. (2013). Accelerating birch for clustering large scale streaming data using cuda dynamic parallelism. In *Intelligent Data Engineering and Automated Learning–IDEAL 2013*, 409–416, Springer. 15

Dong, X.L. & Srivastava, D. (2013). Big data integration. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, 1245 1248, IEEE. 10

Ester, M., Kriegel, H.P., Sander, J. & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, vol. 96, 226–231. 15

Everitt, B.S., Landau, S., Leese, M. & Stahl, D. (2011). *Cluster Analysis, 5th Edition*, i xv. John Wiley Sons Ltd. 6

Fahad, A., Alshatri, N., Tari, Z., Alamri, A., Khalil, I., Zomaya, A.Y., Foufou, S. & Bouras, A. (2014). A survey of clustering algorithms

for big data: Taxonomy and empirical analysis. *Emerging Topics in Computing, IEEE Transactions on*, **2**, 267–279. 13, 44

Fowlkes, E.B. & Mallows, C.L. (1983). A method for comparing two hierarchical clusterings. *Journal of the American statistical association*, **78**, 553–569. 60

Fukunaga, K. & Hostetler, L.D. (1975). The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, **21**, 32–40. 8

Furtado, L., Dutra, M. & Macedo, D. (2017). Value creation in big data scenarios: A literature survey. *Journal of Industrial Integration and Management*, **2**, 1750002. 12

Gan, G., Ma, C. & Wu, J. (2007). *Data clustering: theory, algorithms, and applications*, vol. 20. Siam. 6

Gandomi, A. & Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, **35**, 137–144. 2, 10, 12

Ganti, V., Gehrke, J. & Ramakrishnan, R. (1999). Cactus clustering categorical data using summaries. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 73–83, ACM. 9

Garcia, S. & Herrera, F. (2008). An extension on"statistical comparisons of classifiers over multiple data sets"for all pairwise comparisons. *Journal of Machine Learning Research*, **9**, 2677–2694. 55

Guha, S., Rastogi, R. & Shim, K. (1998). Cure: an efficient clustering algorithm for large databases. In *ACM SIGMOD Record*, vol. 27, 73–84, ACM. 8

Guha, S., Rastogi, R. & Shim, K. (1999). Rock: A robust clustering algorithm for categorical attributes. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, 512–521, IEEE. 8

Halkidi, M., Batistakis, Y. & Vazirgiannis, M. (2001). On clustering validation techniques. *Journal of Intelligent Information Systems*, **17**, 107–145. 9

Halkidi, M., Batistakis, Y. & Vazirgiannis, M. (2002a). Cluster validity methods: Part i. *SIGMOD Rec.*, **31**, 40–45. 9

Halkidi, M., Batistakis, Y. & Vazirgiannis, M. (2002b). Cluster validity methods: part i. *ACM Sigmod Record*, **31**, 40–45. 41

Halkidi, M., Batistakis, Y. & Vazirgiannis, M. (2002c). Clustering validity checking methods: Part ii. *SIGMOD Rec.*, **31**, 19–27. 9

Han, J., Kamber, M. & Pei, J. (2011). *Data mining: concepts and techniques: concepts and techniques*. Elsevier. 6

Harris, M. (2015). Optimizing parallel reduction in cuda. *Nvidia Site*. 30

Hashem, I.A.T., Yaqoob, I., Anuar, N.B., Mokhtar, S., Gani, A. & Khan, S.U. (2015). The rise of big data on cloud computing: review and open research issues. *Information Systems*, **47**, 98–115. 1, 10, 11

Havens, T.C., Bezdek, J.C., Leckie, C., Hall, L.O. & Palaniswami, M. (2012). Fuzzy c-means algorithms for very large data. *IEEE Transactions on Fuzzy Systems*, **20**, 1130–1146. 21

Hesabi, R.Z., Tari, Z., Goscinski, A., Fahad, A., Khalil, I. & Queiroz, C. (2015). *Data Summarization Techniques for Big Data—A Survey*, 1109–1152. Springer New York, New York, NY. 13

Hinneburg, A. & Keim, D.A. (1998). An efficient approach to clustering in large multimedia databases with noise. In *KDD*, vol. 98, 58–65. 13

Hinneburg, A. & Keim, D.A. (1999). Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, 506–517, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 13

Hitzler, P. & Janowicz, K. (2013). Linked data, big data, and the 4th paradigm. *Semantic Web*, **4**, 233 235. 10

Huang, Z. (1997). A fast clustering algorithm to cluster very large categorical data sets in data mining. In *DMKD*, 0, Citeseer. 8

Huang, Z. & Ng, M.K. (1999). A fuzzy k-modes algorithm for clustering categorical data. *Fuzzy Systems, IEEE Transactions on*, **7**, 446–452. 9

Hubert, L. & Arabie, P. (1985). Comparing partitions. *Journal of classification*, **2**, 193–218. 41

Information, O. (2015). Understanding the 7 vs of big data. 10

Ito, K. & Xiong, K. (2000). Gaussian filters for nonlinear filtering problems. *IEEE transactions on automatic control*, **45**, 910–927. 40

Jaccard, P. (1908). Nouvelles recherches sur la distribution florale. *Bulletin de la Sociète Vaudense des Sciences Naturelles*, **44**, 223–270. 41, 60

John Walker, S. (2014). Big data: A revolution that will transform how we live, work, and think. 2

Karypis, G., Han, E.H. & Kumar, V. (1999). Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, **32**, 68–75. 8

Keim, D., Qu, H. & Ma, K.L. (2013). Big data visualization. *Computer Graphics and Applications, IEEE*, **33**, 20 21. 12

Kirk, D. & Hwu, W. (2016). *Programming Massively Parallel Processors: A Hands-on Approach*. Elsevier Science. xi, 14, 15, 16, 17, 18, 19

Kleiner, A., Talwalkar, A., Sarkar, P. & Jordan, M.I. (2014). A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **76**, 795–816. 45

Krishnan, K. (2013). *Data warehousing in the age of big data*. Newnes. 10

Kumari, S., Maheshwari, A., Goyal, P. & Goyal, N. (2015). Parallel framework for efficient k-means clustering. In *Proceedings of the 8th Annual ACM India Conference*, Compute '15, 63–71, ACM, New York, NY, USA. 14

Labrinidis, A. & Jagadish, H.V. (2012). Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, **5**, 2032–2033. 10

Lanczos, C. (1950). *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office. 25, 26

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**, 2278–2324. 58

Leisch, F. & Dimitriadou, E. (2010). mlbench: Machine learning benchmark problems. R package version 2.1-1. 56

Leskovec, J., Rajaraman, A. & Ullman, J.D. (2014). *Mining of massive datasets*. Cambridge university press. 2

Li, F. & Nath, S. (2014). Scalable data summarization on big data. *Distributed and Parallel Databases*, **32**, 313–314. 45

Li, L., Hu, Y. & Wang, X. (2012). A parallel way for computing eigenvector sensitivity of asymmetric damped systems with distinct and repeated eigenvalues. *Mechanical Systems and Signal Processing*, **30**, 61–77. 24

Liang, F., Cheng, Y., Song, Q., Park, J. & Yang, P. (2013). A resampling-based stochastic approximation method for analysis of large geostatistical data. *Journal of the American Statistical Association*, **108**, 325–339. 45

Lim, J.S. (1990). Two-dimensional signal and image processing. *Englewood Cliffs, NJ, Prentice Hall, 1990, 710 p.*. 40

Lin, D. *et al.* (1998). An information-theoretic definition of similarity. In *Icml*, vol. 98, 296–304, Citeseer. 7

Lin, F. & Cohen, W.W. (2010). Power iteration clustering. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, 655–662. 15, 24, 26, 27

MacQueen, J. *et al.* (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, 281–297, Oakland, CA, USA. 8

McAfee, A., Brynjolfsson, E., Davenport, T.H., Patil, D. & Barton, D. (2012). Big data: the management revolution. *Harvard business review*, **90**, 60–68. 11

Meilă, M. (2003). Comparing clusterings by the variation of information. In *Learning theory and kernel machines*, 173–187, Springer. 41

Meila, M. & Shi, J. (2001). A random walks view of spectral segmentation. 24

Ng, A.Y., Jordan, M.I., Weiss, Y. *et al.* (2002). On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, **2**, 849–856. 9

Ng, R.T. & Han, J. (2002). Clarans: A method for clustering objects for spatial data mining. *Knowledge and Data Engineering, IEEE Transactions on*, **14**, 1003–1016. 8

NVIDIA (2016). Nvidia visual profiler tool. 34

O Reilly Media, I. (2015). *Big Data Now: 2014 Edition*, vol. 1. O Reilly Media. 10

O'Leary, D. (2013). Artificial intelligence and big data. *Intelligent Systems, IEEE*, **28**, 96–99. 11

Park, H.S. & Jun, C.H. (2009). A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, **36**, 3336–3341. 8

Parker, J.K. & Hall, L.O. (2014). Accelerating fuzzy-c means using an estimated subsample size. *IEEE Transactions on Fuzzy Systems*, **22**, 1229–1244. 21

Parsons, L., Haque, E. & Liu, H. (2004). Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, **6**, 90–105. 13

Pedrycz, W. & Waletzky, J. (1997). Fuzzy clustering with partial supervision. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **27**, 787–795. 45

R Core Team (2017). R: A language and environment for statistical computing. 55

Rajen Bhatt, A.D. (2012). Skin data set. https://archive.ics.uci.edu/ml/machine-learning-databases/00229/, online; accessed 16-August-2017. 58

Ranjan, R. (2014). Streaming big data processing in datacenter clouds. *IEEE Cloud Computing*, **1**, 78–83. 11

Reeve, A. (2013). *Managing Data in Motion: Data Integration Best Practice Techniques and Technologies*. Newnes. 10

Reiss, A. (2012). Pamap2 physical activity monitoring data set. http://archive.ics.uci.edu/ml/datasets/pamap2+physical+activity+monitoring, online; accessed 16-August-2017. 58

Rendón, E., Abundez, I.M., Gutierrez, C., Zagal, S.D., Arizmendi, A., Quiroz, E.M. & Arzate, H.E. (2011). A comparison of internal and external cluster validation indexes. In *Proceedings of the 2011 American Conference on Applied Mathematics and the 5th WSEAS International Conference on Computer Engineering and Applications*, AMERICAN-MATH'11/CEA'11, 158–163, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA. 9

Rendón, E., Abundez, I.M., Gutierrez, C., Zagal, S.D., Arizmendi, A., Quiroz, E.M. & Arzate, H.E. (2011). A comparison of internal and external cluster validation indexes. In *Proceedings of the 2011 American Conference, San Francisco, CA, USA*, vol. 29. 41

Repository, U.M.L. (2017a). Breast cancer. 56

Repository, U.M.L. (2017b). Iris. 56

Rousseeuw, P.J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, **20**, 53–65.

Royston, P. (1992). Approximating the shapiro-wilk w-test for non-normality. *Statistics and Computing*, **2**, 117–119. 41

Saldaña, J. (2015). *The coding manual for qualitative researchers*. Sage. 21

Sänger, J., Richthammer, C., Hassan, S. & Pernul, G. (2014). Trust and big data: a roadmap for research. In *Database and Expert Systems Applications (DEXA), 2014 25th International Workshop on*, 278–282, IEEE. 12

Shirkhorshidi, A., Aghabozorgi, S., Wah, T. & Herawan, T. (2014). Big data clustering: A review. In B. Murgante, S. Misra, A. Rocha, C. Torre, J. Rocha, M. FalcÃ£o, D. Taniar, B. Apduhan & O. Gervasi, eds., *Computational Science and Its Applications ICCSA 2014*, vol. 8583 of *Lecture Notes in Computer Science*, 707–720, Springer International Publishing. 13, 14, 23

Simon, P. (2014). *The visual organization: data visualization, Big Data, and the quest for better decisions*. John Wiley & Sons. 12

Sivarajah, U., Kamal, M.M., Irani, Z. & Weerakkody, V. (2017). Critical analysis of big data challenges and analytical methods. *Journal of Business Research*, **70**, 263–286. 10

Stetco, A., Zeng, X.J. & Keane, J. (2015). Fuzzy c-means++: Fuzzy c-means with effective seeding initialization. *Expert Systems with Applications*, **42**, 7541 – 7548. 55, 56

Tien, N.D. *et al.* (2017). Tune up fuzzy c-means for big data: some novel hybrid clustering algorithms based on initial selection and incremental clustering. *International Journal of Fuzzy Systems*, **19**, 1585–1602. 22

Tsai, C.W., Lai, C.F., Chao, H.C. & Vasilakos, A.V. (2015). Big data analytics: a survey. *Journal of Big Data*, **2**, 21. 2

Van Rijmenam, M. (2014). *Think Bigger: Developing a Successful Big Data Strategy for Your Business*. AMACOM Div American Mgmt Assn. 10

Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, **17**, 395–416. 24

Wilt, N. (2013). *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Addison-Wesley. 17

Wu, R., Zhang, B. & Hsu, M. (2009). Clustering billions of data points using gpus. In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, 1–6, ACM. 15

Wu, X., Zhu, X., Wu, G.Q. & Ding, W. (2014). Data mining with big data. *Knowledge and Data Engineering, IEEE Transactions on*, **26**, 97–107. 2

Zhang, T., Ramakrishnan, R. & Livny, M. (1996). Birch: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Montreal, Canada*. 8, 13, 15

Zikopoulos, P., Eaton, C. *et al.* (2011). *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media. 10