TÉCNICO LISBOA

Efficient Evidence Accumulation Clustering for large datasets/big data

Diogo Alexandre Oliveira Silva

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Ana Fred 1 and Helena Aidos 2

Examination Committee

Chairperson:	Professor Full Name
Supervisor:	Professor Full Name 1 (or 2)
Member of the Committee:	Professor Full Name 3

Month 2015

Dedicated to someone special...

Acknowledgments

A few words about the university, financial support, research advisor, dissertation readers, faculty or other professors, lab mates, other friends and family...

Resumo

Inserir o resumo em Português aqui com o máximo de 250 palavras e acompanhado de 4 a 6 palavras-chave...

Palavras-chave: palavra-chave1, palavra-chave2,...

Abstract

Insert your abstract here with a maximum of 250 words, followed by 4 to 6 keywords...

Keywords: keyword1, keyword2,...

Contents

List of Tables

List of Figures

Glossary

API	Application Programming Interface.
CPU	Central Processing Unit.
EAC	Evidence Accumulation Clustering.
GPGPU	General Purpose computing in Graphics Processing Units.
GPU	Graphics Processing Unit.
HAC	Hierarchical Agglomeration Clustering.
PCA	Principal Component Analysis.
PC	Principal Component.
QK-Means	Quantum K-Means.
Qubit	Quantum bit.
SL-HAC	Single-Linkage Hierarchical Agglomeration Clustering.
SVD	Singular Value Decomposition.

Chapter 1

Introduction

1.1 The problem of clustering

Advances in technology allow for the collection and storage unprecedented amount and variety of data. Since data is mostly stored electronically, it presents a potential for automatic analysis and thus creation of information and knowledge. A growing body of statistical methods aiming to model, structure and/or classify data already exist, e.g. linear regression, principal component analysis, cluster analysis, support vector machines, neural networks. Many of these methods fall into the realm of machine learning, which is usually divided into 2 major groups: *supervised* and *unsupervised* learning. Supervised learning deals with labelled data, i.e. data for which ground truth is known, and tries to solve the problem of classification. Unsupervised learning deals with unlabelled data and tries to solve the problem of clustering.

Cluster analysis is the backbone of the present work. The goal of data clustering, as defined by [Jain, 2010], is the discovery of the *natural grouping(s)* of a set of patterns, points or objects. In other words, the goal of data clustering is to discover structure on data, structured or not. And the methodology used is to group patterns that are similar by some metric (e.g. euclidean distance, Pearson correlation) and separate those that are dissimilar.

As an example, Figure 1.1 shows the plot of a simple synthetic dataset - a Gaussian mixture of 5 distributions. No extra information other than the position of the points is given, since clustering algorithms are unsupervised methods. Figure 1.2 presents the desired (or ”natural”) clustering for this given dataset. Figure 1.3 presents the clusters given by the K-Means algorithm with an initialization of 4 clusters. The number of clusters was purposefully set to an ”incorrect” number to demonstrate that the number of cluster of a dataset is not trivial to discover, even in suck a simple example. In this synthetic dataset, the number of clusters is not clear due to the two superimposed Gaussians.

The number of clusters is a common initialization parameter for clustering methods. When no prior information about the dataset is given, the number of clusters can be hard to discover.

Cluster analysis is a relevant technique across several domains ([Aggarwal and Reddy]):

- grouping users with similar behaviour or preferences in **customer segmentation**;

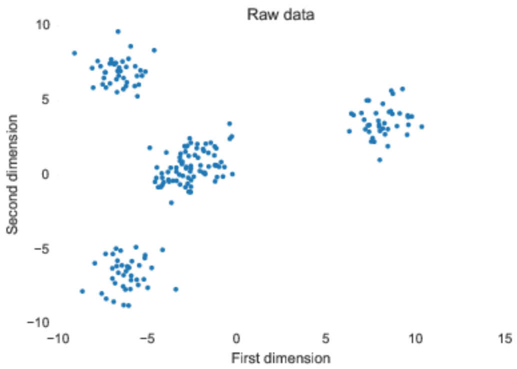


Figure 1.1: Gaussian mixture of 5 distributions. The middle ”ball” of points is 2 Gaussians that intersect.

Figure 1.2: Gaussian mixture of 5 distributions. The colors of each point represents the group (the Gaussian distribution) to which it belongs.

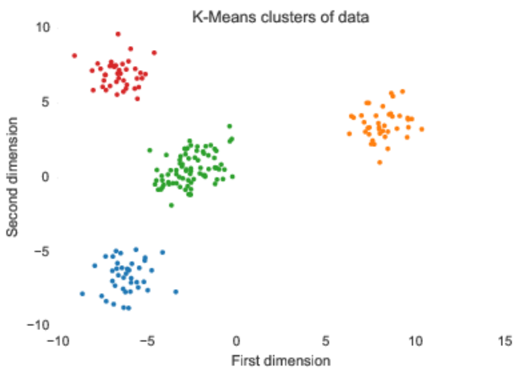


Figure 1.3: Sama data, as Figure 1.1, but the group to which each point belongs to was computed by the K-Means algorithm with the number of clusters set to 4.

- image segmentation in the field of **image processing**;

- clustering gene expression data, among other application, in the domain of **biological data analysis**;

- generation of hierarchical structure for easy access and retrieval of **information systems**;

1.2 Motivation

The scope of the thesis is Big Data and Cluster Ensembles.

Success of EAC clustering on hard data sets.

Interesting problems from big data.

Combining the two.

1.3 Challenges and Motivation

A vast body of work on clustering algorithms exist. Yet, no single algorithm is able to respond to the specificities of all datasets. Different methods are suited to datasets of different characteristics and many times the challenge of the researcher is to find the right algorithm for the task. This is where ensemble methods enter. Ensemble methods use the clustering of several algorithms or several runs of the same algorithm to produce an end result better than any of the individual ones.

This, however, comes at a price. Instead of a single run of an algorithm, several are required, which increases computational complexity. Memory complexity is another aspect to have into account is another aspect to take into account when using such algorithms.

With the rapid increase of storage capacity and the an equal increase on the capability to capture information, the concept of Big Data was borne. There is an interest in clustering much of this data. However, processing such huge amounts of data has been out of the range of capability of the traditional desktop workstation. This sprouted the rise of new computing architectures (e.g. Graphic Processing Units) and programming models (e.g. Hadoop, shared and distributed memory). Each of these has its own specificities and the programmer must have an in-depth knowledge of the architectures and models used to obtain results.

The algorithms themselves are no longer the only focus of research. Much effort is being put into the scalability and performance of algorithms, which usually translates in addressing their memory and computational complexities with parallized computation and distributed memory. This effort comes with challenges. How can one keep the original accuracy while significantly increase efficiency? Is there an explorable trade-off between accuracy and performance that researchers can tap into?

1.4 Goals and Contribution

This dissertation aims to research and extend the state of the art of ensemble clustering, in what concerns the method of Evidence Accumulation Clustering and its application in large datasets, while also accessing alternative algorithmic solutions and parallelization techniques. The goal is to understand EAC's suitability for large datasets and find ways to respond to the challenges that that entails, in terms of speed and memory. In particular, an efficient parallel version for GPU of different parts of the method.

Throughout this dissertation, various clustering techniques are reviewed, implemented and tested.

1.5 Objectives

The main objectives for this work are:

- Review quantum inspired clustering methods
- Study possibility of integration of quantum inspired methods in EAC
- Review of scalability of EAC
- Review methods and techniques designed for processing large datasets
- Review of acceleration techniques for large datasets
- Review of the General Purpose computing in Graphics Processing Units paradigm
- Study possibility of integration of GPGPU in EAC
- Devise strategies to reduce complexity of EAC
- **Application of Evidence Accumulation Clustering in Big Data**
- Validation of Big Data EAC on real data (ECG for emotional state discovery and/or discovery of natural groups)

1.6 Outline

The present document has the following strucutre:

Chapter 2

This chapter starts by reviewing the Evidence Accumulation Clustering algorithm in detail. It goes on to review possible approaches to the problem of scaling EAC. Based on an algorithmic approach, a review of the young field of quantum clustering is presented, with a more in-depth emphasis on two algorithms. With a parallelization approach in mind, a programming model for the GPU (CUDA) is reviewed. Naturally following are some parallelized versions of relevant algorithms to the problem of this dissertation.

Chapter 3

This chapter presents the approach that was actually taken to scale EAC. It presents the steps taken on each part of the algorithm, the underlying difficulties and what was done to address them. It also includes the reference of approaches that are not suited to solve the problem.

Chapter 4

In this section the, the results of the different approaches are presented.

Chapter 5

In this section the, the results are interpreted and discussed. A critical evaluation of the results is offered.

Chapter 6

This chapter concludes the dissertation. It also offers recommendations for future work.

Chapter 2

State of the art

2.1 Quantum clustering

The field of quantum clustering has shown promising results regarding potential speedups in several tasks over their classical counterparts. There are two major paths for the problem of quantum clustering. The first is the quantization of clustering methods to work in quantum computers. This translates in converting algorithms to work partially or totally on a different computing paradigm, with support of quantum circuits or quantum computers. Literature suggests that quadratic (and even exponential in some cases) speedup may be achieved. Most of the approaches for such conversions make use of Groover's search algorithm, or a variant of it, e.g. Wiebe et al. [2014]. Most literature on this path is also mostly theoretical since quantum circuits are not easily available and a working quantum computer has yet to be invented. This path can be seen as part of the bigger problem of quantum computing and quantum information processing.

An alternative to using real quantum systems would be to simulate them. However, simulating quantum systems in classical computers is a very hard task by itself and literature suggest is not feasible. Given that the scope of the thesis is to accelerate clustering, having the extra overhead of simulating the systems would not allow speedups.

The second approach is the computational intelligence approach, i.e. to use algorithms that muster inspiration from quantum analogies. A study of the literature will reveal that this path typically further divides itself into two approaches. One comprehends the algorithms based on the concept of the qubit, the quantum analogue of a classical bit with interesting properties found in quantum objects. The other approach models data as a quantum system and uses the Schrödinger equation to evolve it.

In the following two sections these approaches for quantum inspired computational intelligence are explored.

2.1.1 Quantum bit

To understand the workings of the algorithms based on the concept of the qubit, it is useful to cast some insight about its properties and functioning. The quantum bit is a quantum object that has the properties of quantum superposition, entanglement and ...

A qubit can have any value between 0 and 1 (superposition property) until it is observed, which is when the system collapses to either state. However, the probability with which the system collapses to either state may be different. The superposition property or linear combination of states can be expressed as

$$[\psi] = \alpha[0] + \beta[1]$$

where ψ is an arbitrary state vector and α, β are the the probability amplitude coefficients of basis states $[0]$ and $[1]$, respectively. The basis states correspond to the spin of the modelled particle (in this case, a ferminion, e.g. electron). The coefficients are subjected to the following normalization:

$$|\alpha|^2 + |\beta|^2 = 1$$

where $|\alpha|^2$, $|\beta|^2$ are the probabilities of observing states $[0]$ and $[1]$, respectevely. α and β are complex quantities and represent a qubit:

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Moreover, a qubit string may be represented by:

$$\left[\begin{array}{c|c|c} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \end{array}\right]$$

The probability of observing the state $[000]$ will be $|\alpha_1|^2 \times |\alpha_2|^2 \times |\alpha_3|^2$ To use this model for computing purposes, black-box objects called *oracles* are used.

Def from wiki: In complexity theory and computability theory, an oracle machine is an abstract machine used to study decision problems. It can be visualized as a Turing machine with a black box, called an oracle, which is able to decide certain decision problems in a single operation. The problem can be of any complexity class. Even undecidable problems, like the halting problem, can be used.

2.1.2 Quantum K-Means

Several clustering algorithms Casper and Hung [2013], Casper et al., Xiao et al. [2010], as well as optimization problems Wang et al. [2013], are modelled after this concept. To test the potential of the algorithms under this paradigm, a quantum variant of the K-Means algorithm based on Casper et al. was chosen as a case study.

2.1.3 Description of the algorithm

The Quantum K-Means (QK-Means) algorithm, as is described in Casper et al., is based on the classical K-Means algorithm. It extends the basic K-Means with concepts from quantum mechanics (the qubit) and genetic algorithms.

Within the context of this algorithm, oracles contain strings of qubits and generate their own input by observing the state of the qubits. After collapsing, the qubit value becomes analogue to a classical bit.

Ideally, oracles would contain actual quantum systems or simulate them - this would correctly account for the desirable quantum properties. As it stands, oracles aren't quantum systems or even simulate them. The most appropriate description would be a probabilistic Turing machine. Each string of qubits represents a number, so the number of qubits in each string will define its precision. The number of strings chosen for the oracles depends on the number of clusters and dimensionality of the problem (e.g. for 3 clusters of 2 dimensions, 6 strings will be used since 6 numbers are required). Each oracle will represent a possible solution.

The algorithm has the following steps:

1. Initialize population of oracles
2. Collapse oracles
3. K-Means
4. Compute cluster fitness
5. Store
6. Quantum Rotation Gate
7. Collapse oracles
8. Quantum cross-over and mutation
9. Repeat 3-7 until generation (iteration) limit is reached

Initialize population of oracles The oracles are created in this step and all qubit coefficients are initialized with $\frac{1}{\sqrt{2}}$, so that the system will observe either state with equal probability. This value is chosen taken into account the necessary normalization of the coefficients.

Collapse oracles Collapsing the oracles implies making an observation of each qubit of each qubit string in each oracle. This is done by first choosing a coefficient to use (either can be used), e.g. α. Then, a random value r between 0 and 1 is generated. If $\alpha \geq r$ then the system collapses to $[0]$, otherwise to $[1]$.

K-Means In this step we convert the binary representation of the qubit strings to base 10 and use those values as initial centroids for K-Means. For each oracle, classical K-Means is then executed until it stabilizes or reaches the iteration limit. The solution centroids are returned to the oracles in binary representation.

Compute cluster fitness Cluster fitness is computed using the Davies-Bouldin index for each oracle. The score of each oracle is stored in the oracle itself.

Store The best scoring oracle is stored.

Quantum Rotation Gate So far, we've had classical K-Means with a complex random number generation for the centroids and complicated data structures. This is the step that fundamentally differs from the classical version. In this step a quantum gate (in this case a rotation gate) is applied to all oracles except the best one. The basic idea is to shift the qubit coefficients of the least scoring oracles so they'll have a higher probability of collapsing into initial centroid values closer to the best solution so far. This way, in future generations, we'll not initiate with the best centroids so far (which will not converge further into a better solution) but we'll be closer while still ensuring diversity (which is also a desired property of the genetic computing paradigm). In conclusion, we want to look for better solutions than the one we got before in each oracle while moving in the direction of the best we found so far.

The genetic operations of cross-over and mutation are both part of the genetic algorithms toolbox. Wiebe et al. [2014] suggests that that this operations may not be required to produce variability in the population of qubit strings. This is because, according to Liu et al. [2010], use of the angle-distance rotation method in the quantum rotation operation produces enough variability, with a careful choice of the rotation angle. However, if they were used, their goal is to produce further variability into the population of qubit strings.

2.1.4 Horn and Gottlieb's algorithm

The other approach to clustering that gathers inspiration from quantum mechanical concepts is to use the Schrödinger equation. The algorithm under study was created by Horn and Gottlieb and was later extended by Weinstein and Horn.

The first step in this methodology is to compute a probability density function of the input data. This is done with a Parzen-window estimator in Horn and Gottlieb [2001a], Weinstein and Horn [2009]. The Parzen-window density estimation of the input data is done by associating a Gaussian with each point, such that

$$\psi(\mathbf{x}) = \sum_{i=1}^{N} e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}}$$

where N is the total number of points in the dataset, σ is the variance and ψ is the probability density estimation. ψ is chosen to be the wave function in Schrödinger's equation. The details of why this is are better described in Weinstein and Horn [2009], Horn and Gottlieb [2001a,b].

Having this information we'll compute the potential function $V(x)$ that corresponds to the state of minimum energy (ground state = eigenstate with minimum eigenvalue) Horn and Gottlieb [2001a], by solving the Schrödinger's equation in order of $V(x)$:

$$V(\mathbf{x}) = E + \frac{\frac{\sigma^2}{2}\nabla^2\psi}{\psi} = E - \frac{d}{2} + \frac{1}{2\sigma^2\psi}\sum_{i=1}^{N}\|\mathbf{x}-\mathbf{x}_i\|^2 e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}}$$

And since the energy should be chosen such that ψ is the groundstate (i.e. eigenstate corresponding to minimum eigenvalue) of the Hamiltonian operator associated with Schrödinger's equation (not represented above), the following is true

$$E = -min\frac{\frac{\sigma^2}{2}\nabla^2\psi}{\psi}$$

With all of this, $V(x)$ can be computed. This potential function is akin to the inverse of a probability density function. Minima of the potential correspond to intervals in space where points are together. So minima will naturally correspond to cluster centres Horn and Gottlieb [2001a]. However, it's very computationally intensive to compute $V(x)$ to the whole space, so we only compute the value of this function at the data points. This should not be problematic since clusters' centres are generally close to the data points themselves. Even so, the minima may not lie on the data points themselves. One method to address this problem is to compute the potential on the input data and converge this points toward some minima of the potential function. This is done with the gradient descent method in Horn and Gottlieb [2001a].

Another method Weinstein and Horn [2009] is to think of the input data as particles and use the Hamiltonian operator to evolve the quantum system in the time-dependant Schrödinger equation. Given enough time steps, the particles will converge to and oscillate around potential minima. This method makes the Dynamic Quantum Clustering algorithm. The nature of the computations involved in this algorithm make it a good candidate for parallelization techniques. Wittek [2013] parallelized this algorithm to the GPU obtaining speedups of up to two magnitudes relative to an optimized multicore CPU implementation.

2.2 Evidence Accumulation Clustering

2.2.1 Ensemble Clustering

Ensemble clustering Data from real world problems appear in different configurations regarding shape, size, sparsity, etc. Different clustering algorithms are appropriate for different data configurations, e.g. K-Means using euclidean distance as metric tends to group patterns in hyperspheres so it is more appropriate for data whose structure is formed by hypershere like clusters.If the true structure of the data at hand is heterogeneous in configuration, a single clustering algorithm might perform well for some part of the data while other performs better for some other part. The underlying idea behind ensemble clustering is to use multiple clusterings from one or more clustering algorithms and combine them in such a way that the final clustering is better than any of the individual ones.

Formulation Some notation and nomenclature, adopted from Fred and Jain [2005], should be defined since it will be used throughout the remainder of the present work. The term *data* refers to a set X of n objects or patterns $X = \{\mathsf{x}_1, ..., \mathsf{x}_n\}$, and may be represented by $\chi = \{x_1, ..., x_n\}$, such that $x_i \in \mathbb{R}^d$. A clustering algorithm takes χ as input and returns k groups or *clusters* C of some part of the data, which form a *partition* P. A clustering *ensemble* $\mathbb{P}$ is group of such partitions. This means that:

$$\mathbb{P} = \left\{P^1, P^2, ...P^N\right\} P^j = \left\{\mathsf{x}_1^j, \mathsf{x}_2^j, ...\mathsf{x}_{k_j}^j\right\} C_k^j = \{x_a, x_b, ..., x_z\}$$

2.2.2 overview of EAC

The Evidence Accumulation Clustering (EAC) makes no assumption on the number of clusters in each data partition. Its approach is divided in 3 steps:

1. Produce a clustering ensemble $\mathbb{P}$ (the evidence)
2. Combine the evidence
3. Recover natural clusters

A clustering ensemble, according to Fred and Jain [2005]., can be produced from (1) different data representations, e.g. choice of preprocessing, feature extraction, sampling; or (2) different partitions of the data, e.g. output of different algorithms, varying the initialization parameters.

The ensemble of partitions is combined in the second step, where a non-linear transformation turns the ensemble into a co-association matrix, i.e. a matrix C where each of its elements n_{ij} is the association value between the object pair (i, j). The association between any pair of patterns is given by the number of times those two patterns appear clustered together in any cluster of any partition of the ensemble. The rationale is that pairs that are frequently clustered together are more likely to be representative of a true link between the patterns Fred and Jain [2005], revealing the underlying structure of the data. The construction of this matrix is at the very core of this method.

The co-association matrix itself doesn't output a clustering partition. Instead, it is used as input to other methods to obtain the final partition. Since this matrix is a similarity matrix it's appropriate to use in algorithms take this type of matrices as input, e.g. K-Medoids or hierarchical algorithms such as Single-Link. to name two. Typically, algorithms use a distance as the similarity, which means that they minimize the values of similarity to obtain the highest similarity between objects. However, a low value on the co-association matrix translates in a low similarity between a pair of objects, which means that the co-association matrix requires prior transformation for accurate clustering results, e.g. replace every similarity value n_{ij} between every pair of object (i, j) by $max\{C\} - n_{ij}$.

examples of applications EAC has been used with success in several applications:

- in the field of bioinformatics it was used for the automatic identification of chronic lymphocyt leukemia Qian et al. [2010];

- also in bioinformatics it was used for the unsupervised analysis of ECG-based biometric database to highlight natural groups and gain further insight **?**;
- in computer vision it was used as a solution to the problem of clustering of contour images (from hardware tools) Lourenço and Fred [2007].

advantages

disadvantages quadratic space and time complexities because of the nxn co-association matrix

2.2.3 Scalability of EAC

The quadratic space and time complexity of processing the $n \times n$ co-association matrix is an obstacle to an efficient scaling of EAC. Two approaches have been proposed to address this obstacle: one dealing reducing the co-association matrix by considering only the distances of patterns to theirs p neighbours and the other by maximizing the sparsity of the co-association matrix.

p neighbours approach The first approach, Fred and Jain [2005], proposes an alternative $n \times p$ co-association matrix, where only the p nearest neighbours of each sample are considered in the voting mechanism. This comes at the cost of having to keep track of the neighbours of each pattern in a separate data structure and also of pre-computing the p neighbours. Still, considering that the quadratic space complexity is transformed to $O(2np)$ and that usually $p < \frac{n}{2}$, the cost of storing the extra data structure is lower than that of storing an $n \times n$ matrix, e.g. for a dataset with 10^6 patterns and $p = \sqrt{10^6}$ (a value much higher than that used in Fred and Jain [2005]), the total memory required for the co-association matrix would decrease from $3725.29GB$ to $7.45GB$ (0.18% of the memory occupied by the complete matrix).

Increased sparsity approach The second approach, presented in Lourenço et al. [2010], exploits the sparse nature of the co-association matrix. The co-association matrix is symmetric and with a varying degree of sparsity. The former property translates in the ability of storing only the upper triangular of the matrix without any loss on the quality of the results. The later property is further studied with regards to its relationship with the minimum K_{min} and maximum K_{max} number of clusters in the partitions of the input ensemble. The core of this approach is to only store the non-zero values of the upper triangular of the co-association matrix. The authors study 3 models for the choice of these parameters:

- choice of K_{min} based on the minimum number of gaussians in a gaussian mixture decomposition of the data;
- based on the square root of the number of patterns ($\{K_{min}, K_{max}\} = \{\frac{\sqrt{n}}{2}, \sqrt{n}\}$);
- or based on a linear transformation of the number of patterns ($\{K_{min}, K_{max}\} = \{\frac{n}{A}, \frac{n}{B}\}, A > B$).

The paper compared how each model impacted the sparsity of the co-association matrix (and, thus, the space complexity) and the relative accuracy of the final clusterings. Both theoretical predictions and results revealed that the linear model produces the highest sparsity in the co-association matrix, under an ideal synthetic dataset consisting of a mixture of Gaussians. Furthermore, it is true for both linear and square root models that the sparsity increases as the number of samples increases. In fact, the number non-zero elements is approximately 1000 for each sample in the square root model, whereas it is only 100 in the linear model.

For real datasets, the performance of the three models differed little as the number of samples of the datasets increased. It was found that the chosen granularity of the input partitions (K_{min}) is the variable with most impact, affecting both accuracy and sparsity. The authors reported this technique to have linear space and time complexity on benchmark data.

The number of samples of the datasets analysed in Lourenço et al. [2010] was under $10^(4)$. Although the results appear promising, the present work aims to deal with datasets much larger than this and, as a consequence, this technique should be further evaluated and tested to attest to its usefulness to very large datasets.

2.3 Clustering with Big Data

2.3.1 The concept of Big Data

examples of success application

characteristics and challenges

2.3.2 Computation in Big Data

When big data is in discussion, two perspectives should be taken into account. The first deals with the applications where data is too large to be stored efficiently. This is the problem that streaming algorithms such as X,Y try to solve by analysing data as it is produced, close to real-time processing. The other perspective (the more common) is big data that is actually stored and processed. The latter is the perspective the present work deals with.

Scalability of EAC within the big data paradigm is the concern of this work. Although this line of research hasn't been pursued before, cluster analysis of big data has. Since EAC uses traditional clustering algorithms (e.g. K-Means, Single-Link) in its approach, it is useful to understand how scalable the individual algorithms are as they'll have a big impact in the scalability of EAC. Furthermore, valuable insights may be taken from the techniques used in the scalability of other algorithms.

Clustering algorithms' flow typically involves some initialization step (e.g. choosing k centroids in K-Means) followed by an iterative process until some stopping criteria is met, where each iteration updates the clustering of the data Aggarwal and Reddy. In light of this, to speed up and/or scale up an algorithm, three approaches are available: reduce the number of iterations, reduce the number of patterns the process or parallelizing and distributing the computation. The solutions for each of this approaches are, respectively, one-pass algorithms, randomized techniques that reduce the input space complexity and parallel algorithms.

Parallelization can be attained by adapting programs to multi core CPU, GPU, distributed over several machines (a *cluster*) or a combination of the former, e.g. parallel and distributed processing using GPU in a cluster of hybrid workstations. Each approach has its advantages and disadvantages. The CPU approach has access to a larger memory but the number of computation units is reduced when compared with the GPU or cluster approach. Furthermore CPUs have advanced techniques such as branch prediction, multiple level caching and out of order execution - techniques for optimized sequential computation. GPU have hundreds or thousands of computing units but typically the in device available memory is reduced which entails an increased overhead of memory transfer between host (workstation) and device for computation of large datasets. Furthermore, the scalability cost of these approaches is higher than that of a cluster. Finally, a cluster offers the best solution for truly distributed computation while at the same time allowing for extremely large datasets. The main disadvantage is that there is a high communication and memory I/O cost to pay. Communication is usually done over the network with TCP/IP, which is several order of magnitude slower that the direct access of the CPU or GPU to memory (host or device).

2.4 General Purpose computing on Graphical Processing Units

Using GPU for other applications other than graphic processing, commonly known as GPGPU, has become a trend in recent years. GPU present a solution for ”extreme-scale, cost-effective, and power-efficient high performance computing” Chen and Agrawal [2012]. Furthermore, GPU are typically found in consumer desktops and laptops, effectively bringing this computation power to the masses.

GPUs were typically useful for users that required high performance graphics computation. Other applications were soon explored as users from different fields realized the high parallel computation power of these devices. However, the architecture of the GPUs themselves has been strictly oriented toward the graphics computing until recently as specialized GPU models (e.g. NVIDIA Tesla) have been designed for data computation.

GPGPU application on several fields and algorithms has been reported with significant performance increase, e.g. application on the K-Means algorithm Bai et al. [2009], Wu and Hong [2011], Zechner and Granitzer [2009], Wu et al. [2009], hierarchical clustering Shalom et al. [2009], Arul Shalom and Dash [2011], document clustering Gao et al., image segmentation Sirotkovi et al. [2012], integration in Hadoop clusters Malakar and Vydyanathan [2013], Grossman et al. [2013], among other applications.

Current GPUs pack hundreds of cores and have a better energy/area ratio than traditional infrastructure. GPU work under the SIMD framework, i.e. all the cores in the device execute the same code at the same time and only the data changes over time.

2.4.1 Programming GPUs

In the very beginning of GPGPU, programming was done directly through graphics APIs.

Programming for GPUs was traditionally done within the paradigm of graphics processing, such as DirectX and OpenGL. If researchers and programmers wanted to tap into the computing power of a GPU they had to learn and use these APIs and frameworks, which is a challenging task since their general problems had to be modelled to the graphics-oriented primitives Miši et al. [2012]. With the appearance of DirectX 9, shader programming languages of higher level became available (e.g. C for graphics, DirectX High Level Shader Language, OpenGL Shading Language), but they were still inherently graphics programming languages, where computation must be expressed in graphics terms.

More recent programming models, such as CUDA and OpenCL, removed a lot of that burden by exposing the power of GPUs in a way closer to traditional programming.

At the time of writing, the major programming models used for computation in GPU are OpenCL and CUDA. While the first is widely available in most devices the later is only available for NVIDIA devices. It should also be noted that the MapReduce framework has been implemented on GPU.

It basically boils down to OpenCL vs CUDA. OpenCL has the advantage of portability with the issues of performance portability and har d to program. Programming under CUDA , performs well since it was designed alongside with the hardware itself but only works on NVIDIA devices.

In the end, the choice of GPU computing framework was CUDA. All the infrastructure available for developing and testing supports CUDA. Another reason for the choice is that all the work is being developed in Python and Python has a CUDA API of very high level - part of the Numba module developed by Continuum Analytics.

2.4.2 MapReduce on GPU

As Google's MapReduce computing model has increasingly become a standard for scalable and distributed computing over big data, attempts have been made to port the model to the GPU. This translates in using the same programming model over a wide array of computing platforms.

2.4.3 Overview of CUDA

A GPU is constituted by one or several streaming processors (or multiprocessor). Each of this processors contains several simpler processors, each of which execute a the same instruction at the same time at any given time. In the CUDA programming model, the basic unit of computation is a *thread*. Threads are grouped into *blocks* which are part of the block *grid*. The number of threads in a block is typically higher than the number of processors in a multiprocessor. For that reason, the hardware automatically partitions threads in a block into smaller batches, called *warps*.

Block configuration can be multidimensional, up to and including 3 dimensions. Furthermore, there is a limit to the amount of threads in each dimension that varies with the version of CUDA being used, e.g. for GPUs with CUDA compute capability 2.x the number of threads is 1024 for the x- or y-dimensions, 64 for the z-dimension, an overall maximum number of threads is 1024 and a warp size of 32 threads.

For the previous example, it's wise for the number of threads used in a block to be a multiple of 32 to maximize processor utilization, otherwise some blocks will have processors that will do no work.

Depending on the architecture, GPUs have several types of memories. Accessible to all processors are the global memory, constant memory and texture memory. Blocks share a smaller but significantly faster memory called shared memory. And each thread has it's own, even smaller and faster, local memory.

2.4.4 Parallel K-Means

K-Means is one of the building block of the EAC chain. Other algorithms can be used, but due to its simplicity and speed it is often used to produce ensembles varying the number of centroids to be used. Furthermore, it is a very good candidate for parallelization. K-Means is composed by two main steps:

1. computation of the labels of each pattern in the dataset, e.g. the label of the $n-th$ pattern is 0 if the closest centroid is 0.
2. recomputation of the centroids based on the labels assignment, e.g. the new centroids will be the mean of all the patterns assigned to it.

The first step is inherently parallel as the computation of the label of the $n-th$ pattern is not dependent on any other pattern, but only on the centroids. Two possible approaches to parallelize this step on the GPU are possible, a centroid-centric and a data-centric. In the former each computation unit is responsible for a centroid and will compute the distance from its centroid to every pattern. In the end, the patterns are assigned the closest centroid. This approach is suitable for devices with a low number of computing units so as to stream the data to each one. The later approach is suited to devices with more computing units. Each unit computes the distance from one single pattern to every centroid. In the end, that same unit assigns the pattern to the closest centroid. This strategy has the advantage of using less memory since it doens't need to store all the pair-wise distances to perform the labelling - it only needs to store the best distance for each pattern.

2.4.5 Parallel Hierarchical Agglomerative Clustering

HAC is an algorithm that is not easily parallelized since it has several dependencies

HAC is an important step in the EAC chain. Given the new similarity metric (how many times a pair of patterns are clustered together in the ensemble), HAC provides an intuitive way of obtaining the final partition: patterns that are clustered together often in the ensemble should be clustered together. Furthermore, not knowing the "natural" number of clusters one can use the lifetime criteria, i.e., the number of clusters n should be such that it maximizes the cost of cutting the dendrogram from $n-1$ to n.

However, HAC is not easily parallelized. The most parallelizable part is the computation of the pair-wise similarity matrix, but in EAC is part of the input (it's the co-association matrix) which means that within this context that part is not parallelizable. An important relationship between HAC and MSTs is the key to speeding up HAC, more specifically SL-HAC. When performing SL-HAC, the result is nothing more than a structured MST. To get the n clusters, one cuts the $n-1$ links with highest cost. In this context the nodes of the graph are the patterns of the dataset and the edges are the pair-wise similarities.

Algorithm for finding Minimum Spanning Trees There are several algorithms for computing an MST. The most famous are Kruskal, Prim and Boruvka. The first two are mostly sequential, while the later has the highest potential for parallelization, specially in the first iterations. Boruvka's algorithm is also known as Sollin's algorithm.

Several parallel implementations of this algorithm exist:

Sousa et al. reports their version to be the fastest, as to the moment of writing. The input graph is encoded in the CSR format. This representation is equivalent to having a square matrix G in with zeroed diagonal where the g_{ij} element of the matrix is the weight of the link connecting the node i with the node j. This format is represented in Fig. X.

Within the algorithm's context, the three arrays are *first_edge*, *destination* and *weight*. This three arrays can completely describe a graph. However, the algorithm uses an extra array *outdegree* that can be deduced from the *first_edge* array. The length and purpose of each of this arrays is:

- *first_edge* is an array of size $\|V\|$, where the *i-th* element points to the first edge corresponding to the *i-th* edge.
- *outdegree* is an array of size $\|V\|$, where the *i-th* element contains the number of edges attached to the *i-th* edge.
- *destination* is an array of size $\|E\|$, where the *j-th* element points to the destination vertex of the *j-th* edge.
- *weight* is an array of size $\|E\|$, where the *j-th* element contains the weight of the *j-th* edge.

V is the number of vertices and E is the number of edges. The number of edges is duplicated to cover both directions. The edges in the *destination* array are grouped together by the vertex they originate from, e.g. if edge j is the first edge of vertex i and this vertex has 3 edges, then edges $\{j, j+1, j+2\}$ are the outgoing edges of vertex i.

The algorithm consists on the following steps:

1.

It should be noted, however, that this algorithm doe not support unconnected graphs, i.e., it is not able to output a forest of MSTs. Upon contact, the author reported that a solution to that problem is, on the step of building the flag array, only mark a vertex if it is both the representative of its supervertex and has at least one neighbour.

Exclusive scan The *scan* operation is one of the fundamental building block of parallel computing. Furthermore, two of the steps of the Borůvka variant of Sousa et al. are performed with an exclusive scan where the operation is a sum. To illustrate the functioning of the exclusive scan, let's consider the particular case where the operation of the scan is the sum and the identity (the value for which the operation produces the same output as the input) is naturally 0. Let's further consider the input array to be the sequence $[0, 1, 2, 3, 4, 5, 6, 7]$. Then the output will be $[0, 0, 1, 3, 5, 8, 13, 19]$. The first element of the output will be the identity (if it were an inclusive scan, it would be the first element itself). The second element is the sum between the first element of the input array and the first element of the output array, the third element is the sum between the second element of the input array with second element of the output array, and so on. This algorithm seems highly sequential in nature each element of the output array depends on the previous one. Still, two approaches exist to parallelize it:

- Hillis and Steele
- Bleloch

The two approaches focus on distinct efforts: the former focus on optimizing the number of steps while the later focus on optimizing the amount of work done.

Chapter 3

Methodology

The aim of this thesis is the optimization and scalability of EAC, with a focus for large datasets. EAC is divided in three steps and each has to be considered for optimization.

The first step is the accumulation of evidence, i.e. generating an ensemble of partitions. The main objective for the optimization of this step is speed. Using fast clustering methods for generating partitions is an obvious solution, as is the optimization of particular algorithms aiming for the same objective. Since each partition is independent from every other partition, parallel computing over a cluster of computing units would result in a fast ensemble generation. Using either or any combination of these strategies will guarantee a speedup.

The second step is mostly bound by memory. The complete co-association matrix has a space complexity of $\mathcal{O}(n^2)$. Such complexity becomes prohibitive for big data, e.g. a dataset of 2×10^6 samples will result in a complete co-association matrix of $14901\ GB$ if values are stored in single floating-point precision.

The last step has to take into account both memory and speed requirements. The final clustering must be able to produce good results, fast while not exploding the already big space complexity from the co-association matrix.

Initial research was under the field of quantum clustering. After this pursuit proved fruitless regarding one of the main requirements (computational speed), the focus of researched shifted to parallel computing, more specifically GPGPU.

3.1 Quantum Clustering

Research under this paradigm aimed to find a solution for the first and last steps. Two venues were explored: Quantum K-Means and Horn and Gottlieb's quantum clustering algorithm. For both, the experiments that were setup had the goal of evaluating the speed and accuracy performances of the algorithms.

Under the qubit concept, no other algorithms were experimented with since the results for this particular algorithm showed that this kind of approach is infeasible due to the cost in computational speed.

The results highlight that fact.

3.2 Speeding up Ensemble Generations with Parallel K-Means

K-Means is an obvious candidate for the generation of partitions since it is simple, fast and partitions don't require big accuracy - variability in the partitions is a desirable property which translates in few iterations. For that reason, optimizing this algorithm ensures that the accumulation of evidence is performed in an efficient manner. Furthermore, it is not necessary for K-Means to produce accurate clusterings, e.g. K-Means doesn't have to converge - 3 iterations should suffice. The reason for this is the desire for variability within the partition population.

3.3 Dealing with space complexity of coassocs

3.3.1 Exploiting the sparsity of the co-association matrix

Which method is more effective in very large datasets, however, would depend on the dataset. The sparsity maximization approach got very low densities for some datasets, close to 0.01 in some cases. This is already a big improvement

Either way, the CSR data structure is used to store the co-association matrix. Due to the sparse nature of the co-association matrix, storing it n this format can decrease used space to as much as 10%, depending on the sparsity of the matrix as shown in Lourenço et al. [2010]. This is also an important step since the co-association matrix is already in the correct format for the computation of the final clustering within the GPGPU paradigm.

3.3.2 Using prototypes

k-Nearest Neighbors as prototypes

In the literature review, another approach to reduce the complexity of the co-association matrix is to use the p closest neighbors of each sample. Previous to building the co-association matrix, the p closest samples to each sample are computed and stored in the $n \times p$ neighbor matrix. The co-association matrix is reduced to size $n \times p$ and only considers the neighbors of each sample during the voting mechanism. This means two $n \times p$ matrices have to be stored, which, as long as p is significantly lowers than the number of samples, is close to a sparse representation of the full matrix.

It would be ideal to combine both approaches (sparsity and neighbors) to further reduce space complexity, but they're not necessarily compatible. When the neighbor approach is used, it is unlikely that a sample will never be clustered with it's closest p neighbors. However, this highly depends on the the number of neighbors relative to the number of samples and also the granularity of the partitions. If the number of neighbors is high, one of the neighbors can be sufficiently far away from the sample to not clustered with it. If the granularity of the partitions is high (i.e. there are a high number of small clusters) then each cluster may have sufficiently few samples that neighbors are not included. This means that the $n \times p$ co-association matrix may not have many zeros which translates in little return for using the sparsity augmentation approach. To illustrate this point, let's consider a dataset of 10^6 patterns.

Random prototypes

A second prototype approach is to choose p random non-repetitive samples as prototypes. This will be the same for every sample. Here the voting mechanism is altered so that if a sample is clustered with any of the prototypes, the correspondent element in the co-association matrix is incremented. This has the advantage that only a $n \times p$ matrix needs to be stored along with a p array for the prototypes. Furthermore, if p if high enough to provide a representative sample of the dataset the results can be as good as the full matrix

Medoid prototypes

This approach is similar to the random prototypes but, instead of choosing p random samples from the dataset, the prototypes will be the representatives of the dataset from another algorithm, e.g. K-Medoids, K-Means.

3.4 Hierarchical Aglomerative Clustering step

3.4.1 HAC and GPGPU

The final clustering, done with SL-HAC, is optimized by executing the efficient parallel variant of Borůvka's algorithm Sousa et al. with a slight modification for accepting unconnected graphs, i.e. a co-association matrix where there may be a pattern or a group of patterns that are not connected to any other pattern. In the present implementation, the issue of unconnected graphs was solved in the first step of the efficient variant (finding the minimum vertex connected to each vertex or supervertex). If a vertex has no edges connected to it then it is marked as if its destination points to itself (as a mirrored edge in the second step). This results in unconnected vertices (or supervertices) being dragged throughout all the iterations. As a consequence, the stopping criteria becomes the lack of edges connected to the remaining vertices, which is the same as saying that all elements of the *outdegree* array are zero. This condition can be checked as a secondary result of the computation of the new *first_edge* array. This step is performed by applying an exclusive prefix sum over the *outdegree*. If the prefix sum is implemented in such a way that it returns the sum of all elements, then it becomes inexpensive to check this condition.

The result of this variant is an array of length $|V| - 1$, i.e. $N - 1$ since the vertices are the patterns of the dataset.

-

Chapter 4

Results

4.0.2 Quantum K-Means

The algorithm implemented and tested is a variant of the one described in Casper et al.. The genetic operations of cross-over and mutation are both part of the genetic algorithms toolbox, but were not implemented due to the suggestion from Wiebe et al. [2014]. This decision was based on the findings of Liu et al. [2010], stating that the use of the angle-distance rotation method in the quantum rotation operation produces enough variability, with a careful choice of the rotation angle.

Testing and Results

The testing was aimed at benchmarking both accuracy and speed. The input used was synthetic data, namely, Gaussian mixtures with variable cardinality and dimensionality. The algorithm was implemented in Python 2.7 and the tests were executed in a machine with an Intel i5 processor, 2GB RAM and running Ubuntu 14.04.

(copy of report)

Regarding the Quantum K-Means (QK-Means), the tests were performed using 10 oracles, a qubit string length of 8 and 100 generations per round. The **classical** K-Means was executed using the **k-means++** centroid initialization method, since QK-Means also has some computational cost in the beginning of the algorithm.. Since QK-Means executes a classical K-Means for each oracle each generation, the number of initializations for K-Means was $num.oracles \times num.generations \times factor$, where $factor$ is an adjustable multiplier. Each test had 20 rounds t allow for statistical analysis of the results.

All tests were done with 6 clusters (natural number of clusters). Two tests were done with the two dimensional dataset: one with a $factor = 1.10$ (increase initializations by 10%) and another with $factor = 1$. These tests will be called T1 and T2, respectively. The test done with the six dimensional dataset (T3) used $factor = 1.10$.

Timing results

The mean computation time of classical K-Means is an order of magnitude lower than that of QK-Means. However, in classical K-Means the solution typically chosen is the one with lowest sum of

Table 4.1: Timing results for the different algorithms in the different tests. Fitness time refers to the time that took to compute the DB index of each solution of classical K-Means. All time values are the average over 20 rounds and are displayed in seconds.

Dataset	Algorithm	Mean	Variance	Best	Worst
T1	QK-Means	62.02642975	0.077065212	61.620424	62.579969
bi36	K-Means	6.4774672	0.002501651	6.352554	6.585451
	K-Means + fitness	70.2238286	0.022223755	69.889105	70.548572
	fitness	63.7463614	0.019722105	63.536551	63.963121
T2	QK-Means	64.22347165	0.056559152	63.807367	64.807373
bi36 noFactor	K-Means	5.71167475	0.004903253	5.581391	5.877091
	K-Means + fitness	62.7021533	0.066919692	63.417207	62.180021
	fitness	56.99047855	0.062016439	56.59863	57.540116
T3	QK-Means	74.4917966	0.067688312	74.12105	74.976446
sex36	K-Means	8.291648	0.007015777	8.160859	8.426203
	K-Means + fitness	72.36315915	0.05727269	71.856457	73.031841
	fitness	64.07151115	0.050256913	63.695598	64.605638

squared euclidean distances of points to their attributed centroid. To make a fair comparison between the two algorithms, the Davies-Bouldin index of all classical K-Means solutions was computed and used as the criteria to choose the best solution. When this is done, we can see that the total time of classical K-Means is actually higher that that of QK-Means in T1 and T3, but this is only due to the 1.10 multiplier on the number of initializations. In T2, possibly the fairest comparison, the computation times become very similar with only a 2% difference between the two algorithms.

Accuracy

Comparing K-Means and QK-Means

Table 4.2: All values displayed are the average over 20 rounds, except for the Overall best which shows the best result in any round. The values represent the Davies-Bouldin fitness index (low is better).

Dataset	Algorithm	Best	Worst	Mean	Variance	Overall best
T1	QK-Means	15.42531927	32.29577426	19.94704511	21.23544567	15.42531927
	K-Means	15.42531927	25.44913817	16.25013365	1.216919278	15.42531927
T3	QK-Means	22.72836641	65.19984617	36.10699242	78.14043743	22.71934191
	K-Means	22.71934191	46.72231967	26.18440481	22.96730826	22.71934191

The most relevant result in the table above is the mean of the best index. The value is the average over all rounds of the best solution in each round and it provides insight on the average performance of the algorithm. The results suggest that both algorithms perform equally well. The best overall result of each algorithm in all rounds is exactly the same. In T3, the mean performance of classical K-Means is marginally better.

I speculate that if classical K-Means was using only the sum of euclidean distances and not the DB index, the average performance would be worse. As it stands, choosing to use DB index with classical K-Means possibly represents a tradeoff between speed and accuracy.

QK-Means details

Here we'll analyse a bit what's happening within each QK-Means execution. One would expect for the population's fitness variance to decrease over the generations, as the probabilities for previous known solutions increase and are therefore more likely to reappear. The convergence of the population mean would also be expected to decrease for the same reason. However, experimental (Fig. 4.1 and 4.2) results don't suggest any of these expectations (the results of T1 and T3 suggest the same). This may be due to low number of generations or simply because the random generation of initial centroids isn't influenced enough by the qubit probabilities.

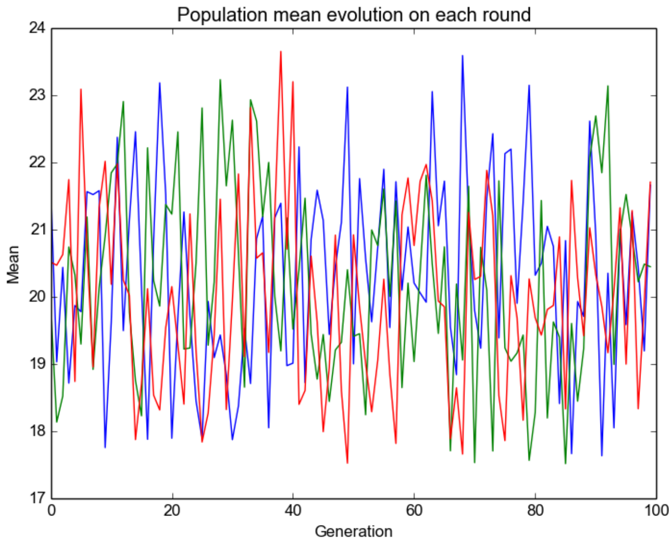


Figure 4.1: DB index mean of the population in T2. Only 4 rounds represented.

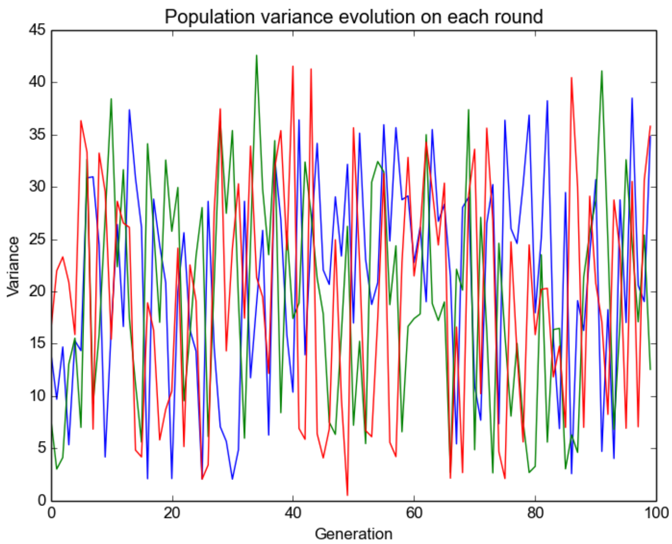


Figure 4.2: DB index variance of the population in T2. Only 4 rounds represented.

Analysing the evolution of the DB index of the best solution over the generations (Fig. 4.3 and 4.4) gives some insight on the rate of convergence. In both tests it is clear that the best solution is often reached in a quarter of the total generations. More detail can be seen in the Table 4.3.

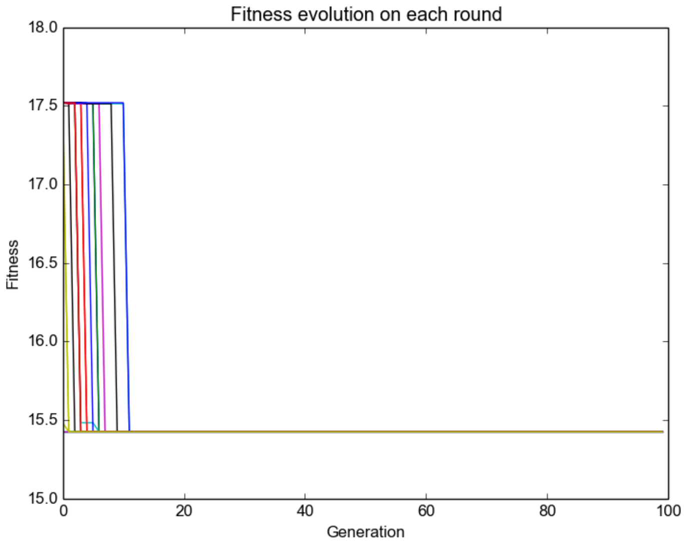


Figure 4.3: DB index of best solution in T2.

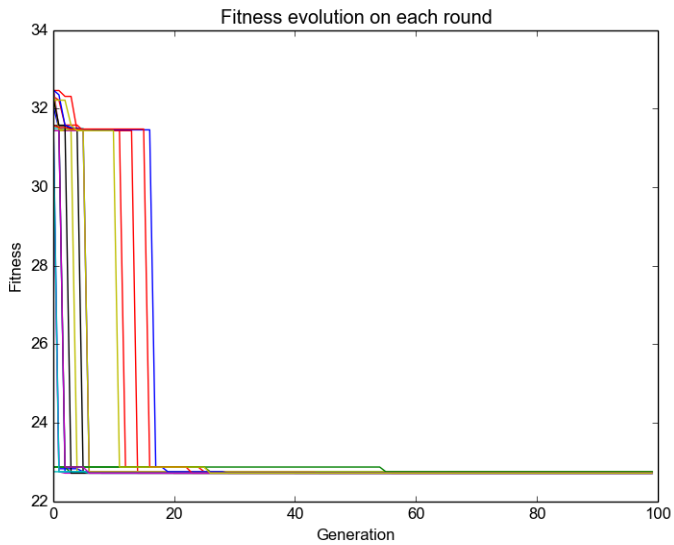


Figure 4.4: DB index of best solution in T3.

Table 4.3: The values represent generations.

Test	Mean	Variance	Best	Worst
T1	17.25	70.2875	3	33
T3	28.05	568.6475	2	90

Discussion

Results show that most computational cost (90% on T1) lies on the evaluation of the solutions obtained from each oracle. This is a costly but necessary step in this algorithm. Moreover, and even though EAC doesn't require its input partitions to be accurate, the quality of the solutions, measured with the Davies-Bouldin index, from QK-Means doesn't differ from that of K-Means. This two facts make the use of this algorithm in EAC prohibitive, as no benefits in compuational time are gained.

It should be noted that the target application of the tests presented differs from that of the original authors and although no accuracy gains were observed in these results, the results might differ on different applications.

4.0.3 Horn and Gottlieb's algorithm

Testing and Results

The accuracy of this algorithm was tested with real world datasets, namely, the crab and iris datasets available at the UCI Machine Learning Repository.

Iris data

The iris dataset ([available at the UCI ML repository](http://archive.ics.uci.edu/ml/datasets/Iris)) has 3 classes each with 50 data points. There are 4 features. The data is preprocessed using Principal Component Analysis (PCA). The natural clustering can be observed in Fig. 4.5.

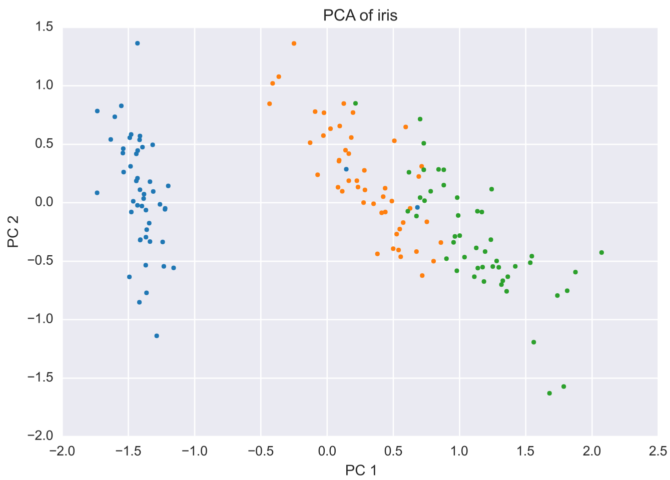


Figure 4.5: Plot of the two first principal components (PC).

I chose $\sigma = \frac{1}{4}$ to reproduce the experiments in [3]. Only the first two PC are used here, which account for 95.8% of the energy. The clustering results can be seen in Fig. 4.6 and have an accuracy of 86% computed with consistency index.

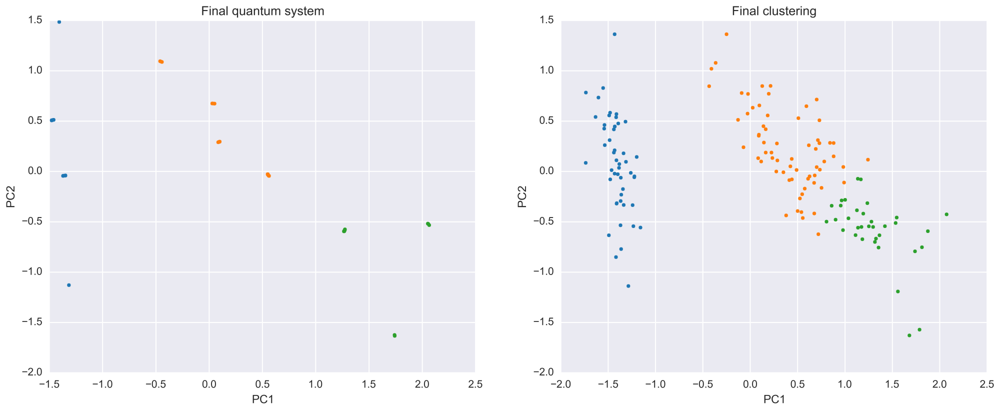


Figure 4.6: Plots of the converged data data points and final clustering for 2 PC.

For the sake of completeness, Fig. 4.7 shows the clustering over all PCs. This solution has an accuracy of 82.67% computed with consistency index.

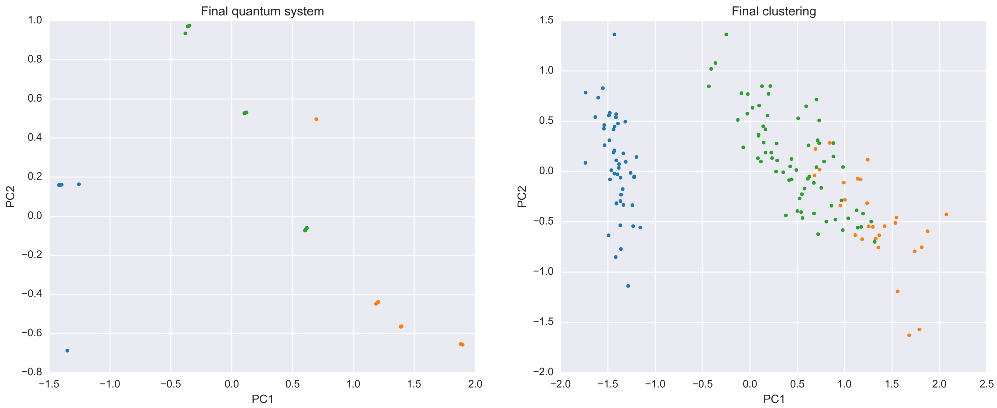


Figure 4.7: Plots of the converged data data points and final clustering for all PC of Iris data.

Crab data

The crabs dataset has 200 samples and describes 5 morphological measurements on 50 crabs each of two colour forms and both sexes (total of 200 crabs), of the species Leptograpsus variegatus collected at Fremantle, Western Australia. After a preprocessing using PCA with covariance matrix and uncentred data, the dataset is represented in Fig. 4.8.

Initial work aimed at reproducing results from [2], but lack of detail on the preprocessing used made it an harder task. Several preprocessings were used, namely whitening or not the data, centring it or not, using covariance versus correlation and different methods of computing the PCs through eigenvalue

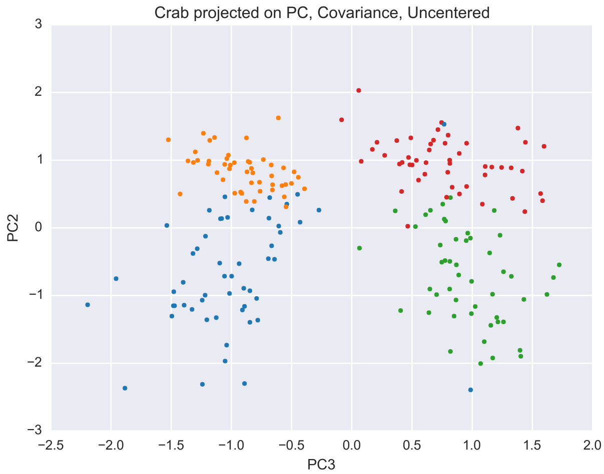


Figure 4.8: Representation of the crab data projected over PC 2 and 3.

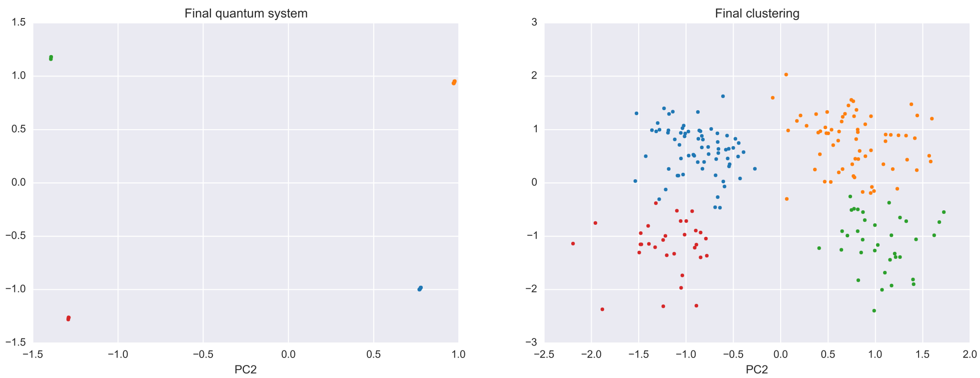


Figure 4.9: Representation of the crab data projected over PC 2 and 3.

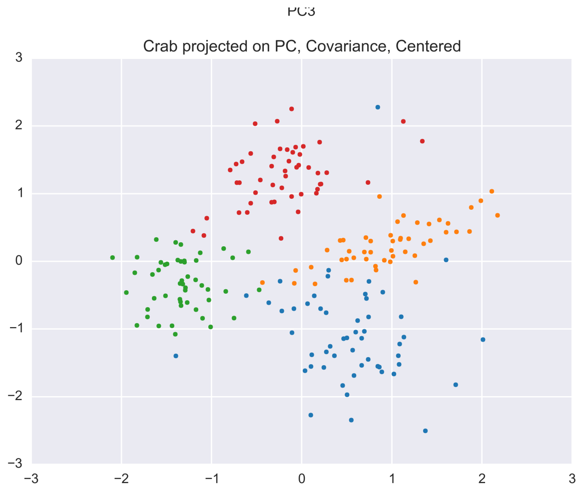
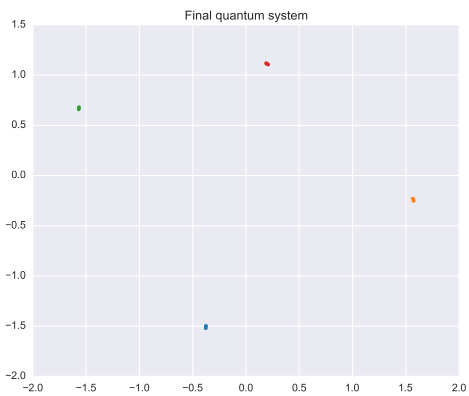


Figure 4.10: Representation of the crab data projected over PC 2 and 3.



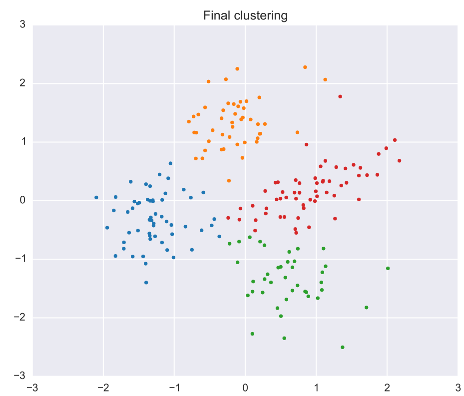


Figure 4.11: Representation of the crab data projected over PC 2 and 3.

decomposition or Singular Value Decomposition (SVD). The closest representation to that of the [2] is the one if Fig. C1.

Covariance uncentred consistency index = 0.815 Covariance centred consistency index = 0.91

all pc covariance uncentred consistency index = 0.63 all dimensions original data consistency index = 0.34

4.0.4 GPGPU K-Means

4.0.5 K-prototypes influence on coassocs

Chapter 5

Discussion?

5.1 Discussion

When a problem of clustering of big data is at hand, the user should reflect upon what the problem at hand really requires: speed or accuracy. The user should take into consideration the nature of the data and the requirements of the problem (concerning speed and accuracy) before proceeding to the execution of the analysis. The present body of work reflects a method of clustering over big data using a high accuracy, but also high cost, method. Other methods offer the opposite, low cost, low to average accuracy.

Chapter 6

Conclusions

Insert your chapter material here...

6.1 Achievements

The major achievements of the present work...

6.2 Future Work

Adaptation of the present implementation to OpenCL. This brings major benefits in respect to portability since OpenCL supports most devices. Moreover, OpenCL's performance is catching in on that of CUDA's and since it's programming model was based on CUDA, it should be straightforward for developers to make the switch.

Application of EAC to the MapReduce framework will further expand the possibilities of application of EAC.

Study the integration of other clustering algorithms within the the EAC toolchain.

Bibliography

C. C. Aggarwal and C. K. Reddy. *Data clustering algorithms and applications*. ISBN 9781466558229.

S. a. Arul Shalom and M. Dash. Efficient hierarchical agglomerative clustering algorithms on GPU using data partitioning. *Parallel and Distributed Computing, Applications and Technologies, PDCAT Proceedings*, pages 134–139, 2011. doi: 10.1109/PDCAT.2011.38.

H. T. Bai, L. L. He, D. T. Ouyang, Z. S. Li, and H. Li. K-means on commodity GPUs with CUDA. *2009 WRI World Congress on Computer Science and Information Engineering, CSIE 2009*, 3:651–655, 2009. doi: 10.1109/CSIE.2009.491.

E. Casper and C.-c. Hung. Quantum Modeled Clustering Algorithms for Image Segmentation 1. 2 (March):1–21, 2013. doi: 10.4156/pica.vol2.issue1.1.

E. Casper, C.-C. Hung, E. Jung, and M. Yang. A Quantum-Modeled K-Means Clustering Algorithm for Multi-band Image Segmentation. URL `http://delivery.acm.org/10.1145/2410000/2401639/p158-casper.pdf?ip=193.136.132.10&id=2401639&acc=ACTIVESERVICE&key=2E5699D25B4FE09E.F7A57B2C5B227641.4D4702B0C3E38B35.4D4702B0C3E38B35&CFID=476955365&CFTOKEN=55494231&__acm__=1423057410_0d77d9b5028cb3`.

L. Chen and G. Agrawal. Optimizing MapReduce for GPUs with Effective Shared Memory Usage. *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing (HPDC'12)*, pages 199–210, 2012. doi: 10.1145/2287076.2287109. URL `http://dl.acm.org/citation.cfm?doid=2287076.2287109$\delimiter"026E30F$nhttp://dl.acm.org/citation.cfm?id=2287109`.

A. N. L. Fred and A. K. Jain. Combining multiple clusterings using evidence accumulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):835–850, 2005.

Z. Gao, E. Li, and Y. Jiang. A gpu-based harmony k-means algorithm for document clustering. pages 2–5.

M. Grossman, M. Breternitz, and V. Sarkar. HadoopCL: MapReduce on distributed heterogeneous platforms through seamless integration of hadoop and OpenCL. *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, IPDPSW 2013*, pages 1918–1927, 2013. doi: 10.1109/IPDPSW.2013.246.

D. Horn and A. Gottlieb. The Method of Quantum Clustering. *NIPS*, (1), 2001a. URL http://www-2.cs.cmu.edu/Groups/NIPS/NIPS2001/papers/psgz/AA08.ps.gz.

D. Horn and A. Gottlieb. Algorithm for Data Clustering in Pattern Recognition Problems Based on Quantum Mechanics. *Physical Review Letters*, 88(1):1–4, 2001b. ISSN 0031-9007. doi: 10.1103/PhysRevLett.88.018702. URL http://journals.aps.org/prl/abstract/10.1103/PhysRevLett.88.018702.

A. K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, 2010. ISSN 01678655. doi: 10.1016/j.patrec.2009.09.011. URL http://dx.doi.org/10.1016/j.patrec.2009.09.011.

W. Liu, H. Chen, Q. Yan, Z. Liu, J. Xu, and Y. Zheng. A novel quantum-inspired evolutionary algorithm based on variable angle-distance rotation. *2010 IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*, 2010. doi: 10.1109/CEC.2010.5586281.

A. Lourenço and A. Fred. Ensemble methods in the clustering of string patterns. *Proceedings - Seventh IEEE Workshop on Applications of Computer Vision, WACV 2005*, pages 143–148, 2007. doi: 10.1109/ACVMOT.2005.46.

A. Lourenço, A. L. N. Fred, and A. K. Jain. On the scalability of evidence accumulation clustering. *Proceedings - International Conference on Pattern Recognition*, 0:782–785, 2010. ISSN 10514651. doi: 10.1109/ICPR.2010.197.

R. Malakar and N. Vydyanathan. A CUDA-enabled hadoop cluster for fast distributed image processing. *2013 National Conference on Parallel Computing Technologies, PARCOMPTECH 2013*, 2013. doi: 10.1109/ParCompTech.2013.6621392.

M. J. Miši, M. Ĉ, and M. V. Tomaševi. Evolution and Trends in GPU Computing. *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 289–294, 2012.

Y. W. Qian, W. Cukierski, M. Osman, and L. Goodell. Combined multiple clusterings on flow cytometry data to automatically identify chronic lymphocytic leukemia. *ICBBT 2010 - 2010 International Conference on Bioinformatics and Biomedical Technology*, pages 305–309, 2010. doi: 10.1109/ICBBT.2010.5478955.

S. a. A. Shalom, M. Dash, M. Tue, and N. Wilson. Hierarchical agglomerative clustering using graphics processor with compute unified device architecture. *2009 International Conference on Signal Processing Systems, ICSPS 2009*, pages 556–561, 2009. doi: 10.1109/ICSPS.2009.167.

J. Sirotkovi, H. Dujmi, and V. Papi. K-Means Image Segmentation on Massively Parallel GPU Architecture. pages 489–494, 2012.

C. d. S. Sousa, A. Mariano, and A. Proença. A Generic and Highly Efficient Parallel Variant of Boruvka 's Algorithm. URL https://github.com/Beatgodes/BoruvkaUMinho.

H. Wang, J. Liu, J. Zhi, and C. Fu. The Improvement of Quantum Genetic Algorithm and Its Application on Function Optimization. 2013(1), 2013.

M. Weinstein and D. Horn. Dynamic quantum clustering: a method for visual exploration of structures in data. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 80(6):1–15, Dec. 2009. ISSN 1539-3755. doi: 10.1103/PhysRevE.80.066117. URL `http://link.aps.org/doi/10.1103/PhysRevE.80.066117`.

N. Wiebe, A. Kapoor, and K. Svore. Quantum Algorithms for Nearest-Neighbor Methods for Supervised and Unsupervised Learning. page 31, 2014. URL `http://arxiv.org/abs/1401.2142`.

P. Wittek. High-performance dynamic quantum clustering on graphics processors. *Journal of Computational Physics*, 233:262–271, 2013. ISSN 00219991. doi: 10.1016/j.jcp.2012.08.048. URL `http://dx.doi.org/10.1016/j.jcp.2012.08.048`.

J. Wu and B. Hong. An efficient k-means algorithm on CUDA. *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 1740–1749, 2011. ISSN 1530-2075. doi: 10.1109/IPDPS.2011.331.

R. Wu, B. Zhang, and M. Hsu. Clustering billions of data points using GPUs. *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, pages 1–5, 2009. doi: 10.1145/1531666.1531668. URL `http://portal.acm.org/citation.cfm?id=1531666.1531668`.

J. Xiao, Y. Yan, J. Zhang, and Y. Tang. A quantum-inspired genetic algorithm for k-means clustering. *Expert Systems with Applications*, 37:4966–4973, 2010. ISSN 09574174. doi: 10.1016/j.eswa.2009.12.017. URL `http://ac.els-cdn.com/S095741740901063X/1-s2.0-S095741740901063X-main.pdf?_tid=f303a76c-ac71-11e4-be73-00000aacb35e&acdnat=1423056793_66291f279193fa69b86c93aecea405b0`.

M. Zechner and M. Granitzer. Accelerating k-means on the graphics processor via CUDA. *Proceedings of the 1st International Conference on Intensive Applications and Services, INTENSIVE 2009*, pages 7–15, 2009. doi: 10.1109/INTENSIVE.2009.19.

Appendix A

Vector calculus

In case an appendix if deemed necessary, the document cannot exceed a total of 100 pages...

Some definitions and vector identities are listed in the section below.

A.1 Vector identities

$$\nabla \times (\nabla \phi) = 0 \tag{A.1}$$

$$\nabla \cdot (\nabla \times \mathbf{u}) = 0 \tag{A.2}$$