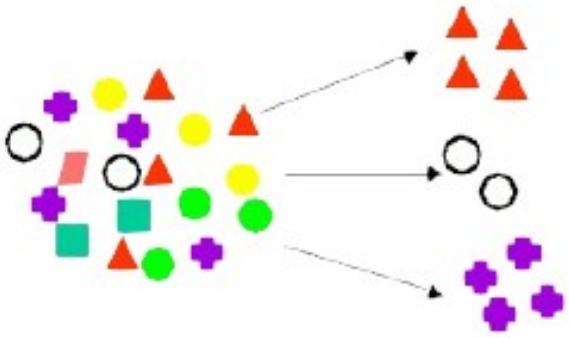


Efficient Evidence Accumulation Clustering for large datasets/big data

Diogo Alexandre Oliveira Silva
Alferes Aluno Engenheiro Electrotécnico 136787-A

Thesis to obtain the Master of Science Degree in

Aeronautical Military Sciences in the speciality of Electrical Engineering

Examination Committee

Chairperson:	Professor Full Name
Supervisor:	Dra. Ana Luísa Nobre Fred
Co-supervisor:	Dra. Helena Aidos Lopes
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

Clustering: basic concepts, definitions and algorithms

Hundreds of methods for data analysis exist. 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 labeled data, i.e. data for which ground truth is known, and tries to solve the problem of classification. Examples of supervised learning algorithms are Neural Networks, Decision Trees, Linear Regression and Support Vector Machines. Unsupervised learning deals with unlabeled data for which no extra information is known. An example of algorithms within this paradigm is clustering algorithms, which are the focus of this chapter.

This chapter will serve as an introduction to clustering. It starts by defining the problem of clustering in section 1.1, goes on to provide useful definitions and notation in section 1.2 and briefly addresses different properties of clustering algorithms in section 1.3. Two very well known algorithms are presented: K-Means in section 1.4 and Single-Link in section 1.5. Evidence Accumulation Clustering is a state of the art ensemble clustering algorithm and the focus of this dissertation. Section 1.6 will explain briefly the concept of ensemble clustering followed by an overview and application examples of the EAC algorithm in section 1.7.

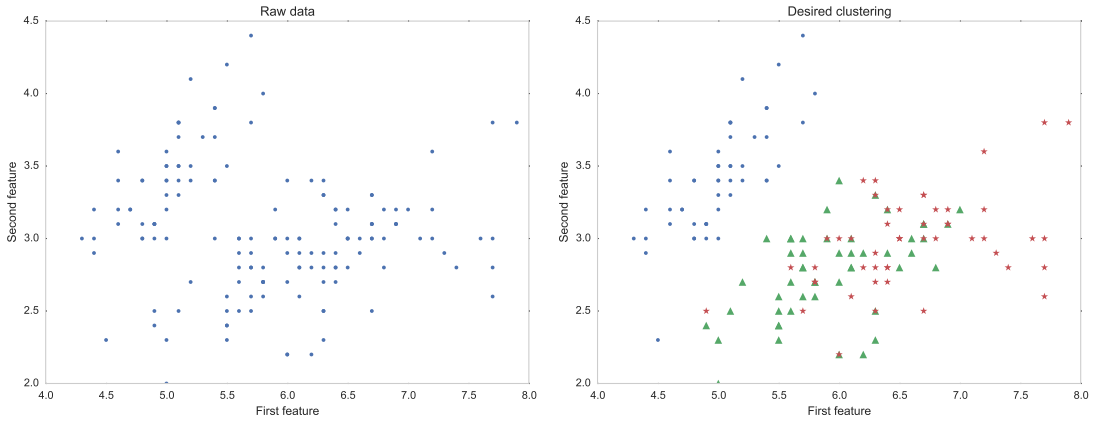
1.1 The problem of clustering

Cluster analysis methods are unsupervised and the backbone of the present work. The goal of data clustering, as defined by [2], 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. The methodology used is to group patterns (usually represented as a vector of measurements or a point in space [3]) based on some similarity, such that patterns belonging to the same cluster are typically more similar to each other than to patterns of other clusters. Clustering is a strictly data-driven method, in contrast with classification techniques which have a training set with the desired labels for a limited collection of patterns. Because there is very little information, as few assumptions as possible should be made

about the structure of the data (e.g. number of clusters). And, because clustering typically makes as few assumptions on the data as possible, it is appropriate to use it on exploratory structural analysis of the data. The process of clustering data has three main stages [3]:

- **Pattern representation** refers to the choice of representation of the input data in terms of size, scale and type of features. The input patterns may be fed directly to the algorithms or undergo *feature selection* and/or *feature extraction*. The former is simply the selection of which features of the originally available should be used. The latter deals with the transformation of the original features such that the resulting features will produce more accurate and insightful clusterings, e.g. Principal Component Analysis.
- **Pattern similarity** refers to the definition of a measure for computing the similarity between two patterns.
- **Grouping** refers to the algorithm that will perform the actual clustering on the dataset with the defined pattern representation, using the appropriate similarity measure.

As an example, Figure 1.1a shows the plot of the Iris data set [4, 5], a small well-known Machine Learning data set. This data set has 4 features, of which only 2 are represented, and 3 classes, of which 2 are overlapping. A class is overlapping another if they share part of the feature space, i.e. there is a zone in the feature space whose patterns might belong to either class. Figure 1.1b presents the desired clustering for this data set.



(a) Input data, unlabeled. (b) Desired labels.

Figure 1.1: First and second features of the Iris dataset. Fig. 1.1a shows the raw input data, i.e. how the algorithms ”see” the data. Fig. 1.1b shows the desired labels for each point, where each color is coded to a class.

1.2 Definitions and Notation

This section will introduce relevant definitions and notation within the clustering context that will be used throughout the rest of this document and were largely adopted from [3].

A *pattern* $\mathbf{x}$ is a single data item and, without loss of generality, can be represented as a vector of d *features* x_i that characterize that data item, $\mathbf{x} = (x_1, \ldots, x_d)$, where d is referred to as the dimensionality of the pattern. A *pattern set* (or data set) $\mathcal{X}$ is then the collection of all n patterns $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. The number of features is usually the same for all patterns in a given pattern set.

In cluster analysis, the desired clustering, typically, is one that reflects the natural structure of the data, i.e. the original ground truth labeling. In other words, one wants to group the patterns that came from the same state of nature when they were generated, the same *class*. A class, then, can be viewed as a source of patterns and the effort of the clustering algorithm is to group patterns from the same source. Throughout this work, these classes will also be referred to as the "natural" or "true" clusterings. *Hard* clustering (or partitional) techniques assign a class label l_i to each pattern $\mathbf{x}_i$. The whole set of labels corresponding to a pattern set $\mathcal{X}$ is given by $\mathcal{L} = \{l_1, \ldots, l_n\}$, where l_i is the label of pattern $\mathbf{x}_i$. Closely related to the whole set of labels is the concept of a *partition*, which completely describes a clustering. A partition P is a collection of k *clusters*. A cluster C is a subset of nc patterns $\mathbf{x}_i$ taken from the pattern set, where the patterns belonging to one subset do not belong to any other in the same partition. A clustering *ensemble* $\mathbb{P}$ is a set N partitions P^j from a given pattern set, each of which is composed by a set of k_j clusters C_i^j, where $j = 1, \ldots, N$, $i = 1, \ldots, k_j$. Each cluster is composed by a set of nc_i^j patterns that does not intercept any other cluster of the same partition. The relationship between the above concepts is condensed in the following expressions:

$$\begin{aligned} ensemble \quad & \mathbb{P} = \left\{P^1, P^2, \ldots P^N\right\} \\ partition \quad & P^j = \left\{C_1^j, C_2^j, \ldots C_{k_j}^j\right\} \\ cluster \quad & C_i^j = \left\{x_1, x_2, \ldots x_{nc_i^j}\right\} \end{aligned}$$

Typically, a clustering algorithm will use a *proximity* measure for determining how alike are two patterns. A proximity measure can either be a *similarity* or a *dissimilarity* measure. One can easily be converted to the other and the main difference is that the former increases in value as patterns are more alike, while the latter decreases in value. A *distance* is a dissimilarity function *d* which yields non-negative real values and is also a *metric*, which means it obeys the following three properties:

$$\begin{aligned} identity \quad & d(\mathbf{x}_i, \mathbf{x}_i) = 0 \\ symmetry \quad & d(\mathbf{x}_i, \mathbf{x}_j) = d(\mathbf{x}_j, \mathbf{x}_i), i \neq j \\ triangle\ inequality \quad & d(\mathbf{x}_i, \mathbf{x}_j) + d(\mathbf{x}_j, \mathbf{x}_z) \geq d(\mathbf{x}_x, \mathbf{x}_z) \end{aligned}$$

where $\mathbf{x}_i$, $\mathbf{x}_j$ and $\mathbf{x}_z$ are 3 unique patterns belonging to the pattern set $\mathcal{X}$. Examples of proximity measures include the Euclidean distance, the Pearson's correlation coefficien and Mutual Shared Neighbors [6]. It should be noted that different proximity measures may be more appropriate in different contexts, such as document, biological or time-series clustering. Furthermore, data can come in different

types such as numerical (discrete or continuous) or categorical (binary or multinomial) attributes. The researcher should take these factors into account as different proximity measures are more appropriate for some type or even heterogeneous type data.

An introduction of clustering would be incomplete without a discussion on how good is a partition after clustering. Several *validation measures* exist and they can placed in two main categories [7]. *External* measures use *a priori* information about the data to evaluate the clustering against some external structure. An application of an external measure could be to test how accurate a clustering algorithm is for a particular dataset by matching the output partition against the ground truth. Examples of such measures include the *Consistency Index* [8] and the H-index [9]. *Internal* measures, on the other hand, determine the quality of the clustering without the use of external information about the data. The Davies-Bouldin index [10] is such a measure.

1.3 Characteristics of clustering techniques

Clustering algorithms may be categorized and described according to different properties. For the sake of completeness, a brief discussion of some of their properties will be layed out in this section.

It is common to organize cluster algorithms into two distinct types: *partitional* and *hierarchical*. A partitional algorithm, such as K-Means, is a hard clustering algorithm that will output a partiton where each pattern belongs exclusively to one cluster. A hierarchical algorithm produces a tree-based data structure called *dendrogram*. A dendrogram contains different partitions at different levels of the tree which means that the user can easily change the desired number of clusters by simply traversing the different levels. This is an advantage over a partitional algorithm since a user can analyze different partitions with different numbers of clusters without having to rerun the algorithm. Hierarchical algorithms can be further split into two approaches: bottom-up (or *agglomerative*) and top-down (or *divisive*). The former starts with all patterns as distinct clusters and will group them together according to some dissimilarity measure, building the dendrogram from the ground up; examples of algorithms that take this approach are Single-Link and Average-Link. The latter will start will all patterns in the same cluster and continuosly split it until all patterns are separated, building the dendrogram from the top level to the bottom; this approach is taken by the Principal Directon Divisive Partitioning[11] and Bisecting K-Means [12] algorithms.

Another characteristic relates to how algorithms use the features for computing similarities. If all features are used simultaniously the algorithm is called *polithetic*, e.g. K-Means. Otherwise, if the features are used sequentially, it is called *monothetic*, e.g. [13].

Contrasting *hard* clustering algorithm are the *fuzzy* algorithms. A fuzzy algorithm will attribute to each pattern a degree of membership to each cluster. A partition can still be extracted from this output by choosing, for each pattern, the cluster with higher degree of membership. An example of a fuzzy algorithm is the Fuzzy C-Means [14].

Another characteristic is an algorithm's stochasticity. A *stochastic* algorithms uses a probabilistic process at some point in the algorithms, possibly yielding different results in each run, e.g. K-Means

can use a random initialization. As an example, the K-Means algorithm typically picks the initialization centroids randomly. A *deterministic* algorithm, on the other hand, will always produce the same result for a given input, e.g. Single-Link.

Finally, the last characteristic discussed is how an algorithm processes the input data. An algorithm is said to be *incremental* if it processes the input incrementally, i.e. taking part of the data, processing it and then doing the same for the remaining parts, e.g. PEGASUS [15]. A *non-incremental* algorithm, on the other hand, will process the whole input in each run, e.g. K-Means. This discussion is specially relevant when considering large datasets that may not fit in memory or whose processing would take too long for a single run and is therefore done in parallel.

1.4 K-Means

One of the most famous non-optimal solutions for the problem of partitional clustering is the K-Means algorithm [16]. The K-Means algorithm uses K *centroid* representatives, c_k, for K clusters. Patterns are assigned to a cluster such that the squared error (or, more accurately, squared dissimilarity measure) between the cluster representatives and the patterns is minimized. In essence, K-Means is a solution (although not necessarily an optimal one) to an optimization problem having the Sum of Squared Errors as its objective function, which is known to be a computationally NP hard problem [2]. It can be mathematically demonstrated that the optimal representatives for the clusters are the means of the patterns of each cluster [7]. K-Means, then, minimizes the following expression, where the proximity measure used is the Euclidean distance:

$$\sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - c_k\|^2 \tag{1.1}$$

K-Means needs two initialization parameters: the number of clusters and the centroid initializations. It starts by assigning each pattern to its closer cluster based on the cluster's centroid. This is called the **labeling** step since one usually uses cluster labels for this assignment. The centroids are then recomputed based on this assignment, in the **update** step. The new centroids are the mean of all the patterns belonging to the clusters, hence the name of the algorithm. These two steps are executed iteratively until a stopping condition is met, usually the number of iterations, a convergence criteria or both. The initial centroids are usually chosen randomly, but other schemes exist to improve the overall accuracy of the algorithm, e.g. K-Means++ [17]. There are also methods to automatically choose the number of clusters [7].

The proximity measure used is typically the Euclidean distance. This tends to produce hyperspherical clusters [3]. Still, according to [2], other measures have been used such as the L1 norm, Mahalanobis distance, as well as the cosine similarity [7]. The choice of similarity measure must be made carefully as it may not guarantee that the algorithm will converge.

A detail of implementation is what to do with clusters that have no patterns assigned to them. One

approach to this situation is to drop the empty clusters in further iterations. However, allowing the existence of empty clusters or dropping empty clusters is undesirable since the number of clusters is an input parameter and it is expected that the output contains the specified number of clusters. Other approaches exist dealing with this problem, such as equaling the centroid of an empty cluster to the pattern furthest away from its assigned centroid or reusing the old centroids as in [18].

K-Means is a simple algorithm with reduced complexity $O(ntk)$, where n is the number of patterns in the pattern set, k is the number of clusters and t is the number of iterations that it executes. Accordingly, K-Means is often used as foundational step of more complex and robust algorithms, such as the EAC algorithm.

As an example, the evolution and output of the K-means algorithm to the data presented in Fig. 1.1 is represented in Fig. 1.2. The algorithm was executed with 3 random centroids.

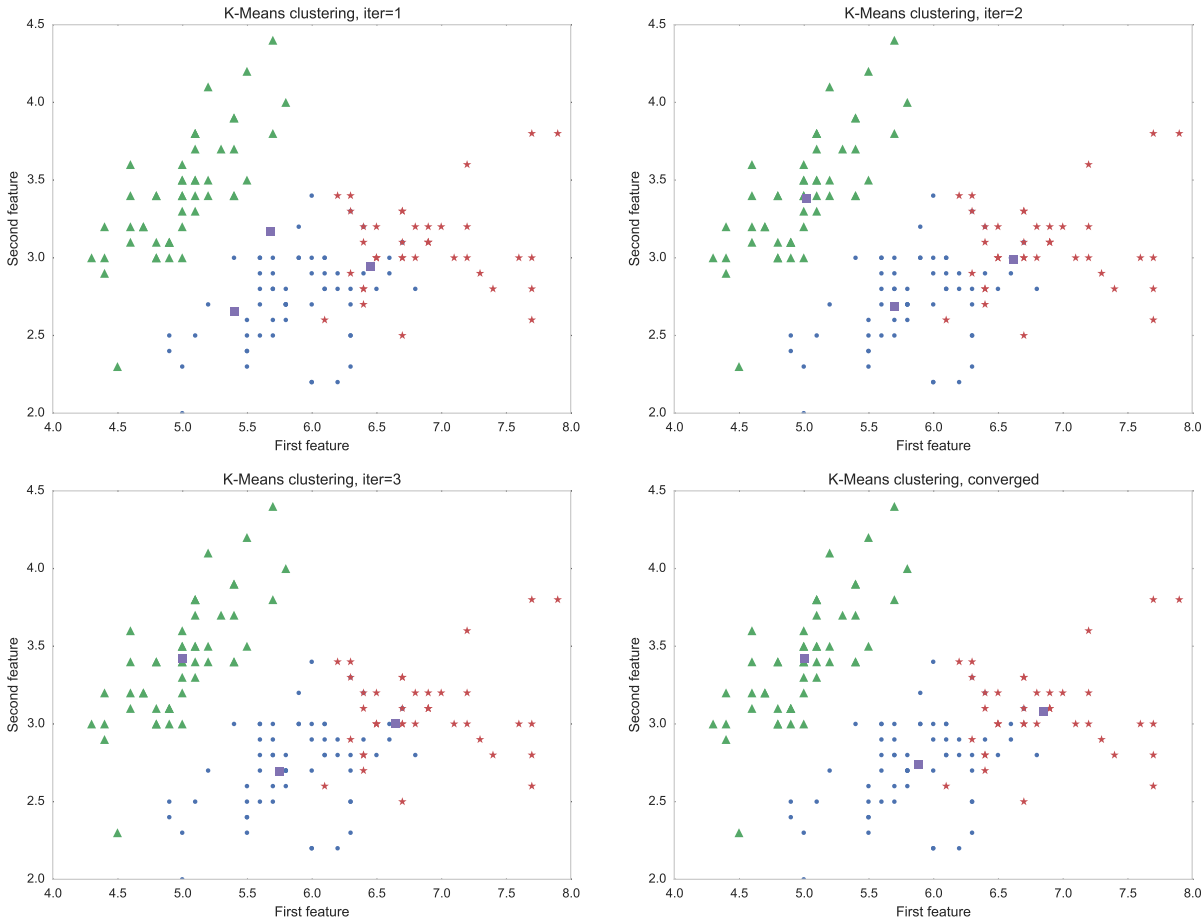


Figure 1.2: The output labels of the K-Means algorithm with the number of clusters (input parameter) set to 3. The different plots show the centroids (squares) evolution on each iteration. Between iteration 3 and the converged state 2 more iterations were executed.

Even with the correct number of clusters, the clustering results do not match 100% the natural clusters. The accuracy relative to the natural clusters of Fig. 1.1b is 88% as measured by the Consistency Index (CI) [8]. In this example, the problem is the two overlapping clusters. It is hard for an algorithm to discriminate between two clusters when they have similar patterns. When no prior information about the dataset is given, the number of clusters can be hard to discover. This is why, when available, a domain

expert may provide valuable insight on tuning the initialization parameters.

1.5 Single-Link

Single-Link [19] is one of the most popular hierarchical agglomerative clustering (HAC) algorithms. HAC algorithms operate over a pair-wise dissimilarity matrix and outputs a dendrogram (e.g. Fig 1.4a). The main steps of an agglomerative hierarchical clustering algorithm are the following [3]:

1. Create a pair-wise dissimilarity matrix of all patterns, where each pattern is a distinct cluster singleton;

2. Find the closest clusters, merge them and update the matrix to reflect this change. The rows and columns of the two merged clusters are deleted and a new row and column are created to store the new cluster.

3. Stop if all patterns belong to a single cluster, otherwise continue to step 2.

The algorithm stops when $n-1$ merges have been performed, which is when all patterns have been connected in the same cluster. Just like in the K-Means algorithm, different similarity measures can be used for the distances.

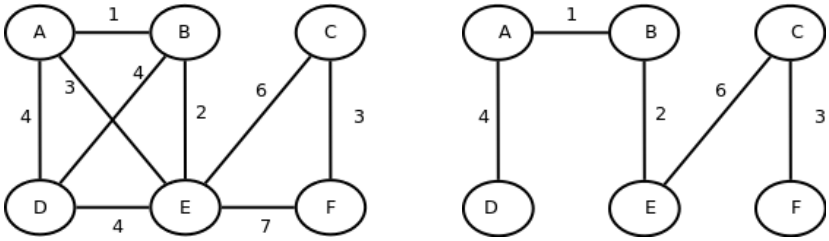
The proximity between clusters in the second step is distinguishes between the different HAC linkage algorithms, such as Single-Link , Average-Link, Complete-Link, among others. In Single-Link (SL), the proximity between any two clusters is the the dissimilarity between their closest patterns. On the other hand, in Complete-Link, it is the proximity between their most distant patterns and, in Average-Link, is the proximity between the average point of each cluster. In SL, because the algorithm connects first clusters that are more similar, it naturally gives more importance to regions of higher density [7].

The total time complexity of a naive implementation is $O(n^3)$ since it performs a $O(n^2)$ search in step two and it does it $n-1$ times. Over time, more efficient implementations have been proposed, such as SLINK [20]. SLINK needs no working copy of $O(n^2)$ the pair-wise similarity matrix (if the original can be modified), has a working memory of $O(n^2)$ and time complexity of $O(n^2)$. This increase in performance comes from the observation that the $O(n^2)$ search can be transformed in a $O(n)$ search at the expense of keeping two arrays of length n that will store the most similar cluster for each pattern and the corresponding similarity measure. This way, to find the two closest clusters, the algorithm will not search the entire similarity matrix, but only the new similarity array since this array keeps the closest cluster of each cluster. Naturally these arrays must be updated upon a cluster merge.

An interesting property of the SL algorithm is its equivalence with a Minimum Spanning Tree (MST), an observation first made by [21]. In graph theory, a MST is a tree that connects all vertices together while minimizing the sum of all the distance between them. An example of a graph and its corresponding MST can be seen in Fig. 1.3. In this context, the edges of the MST are the distances between the patterns and the vertices are the patterns themselves. A MST contains all the information necessary to build a Single-Link dendrogram. To walk down through the levels of the dendrogram from the MST, one

cuts the least similar edges. Furthermore, this approach can be used to apply Single-Link clustering to graphs-encoded problems in a straight-forward way. Furthermore, the performance properties of this method are roughly the same as SLINK [22].

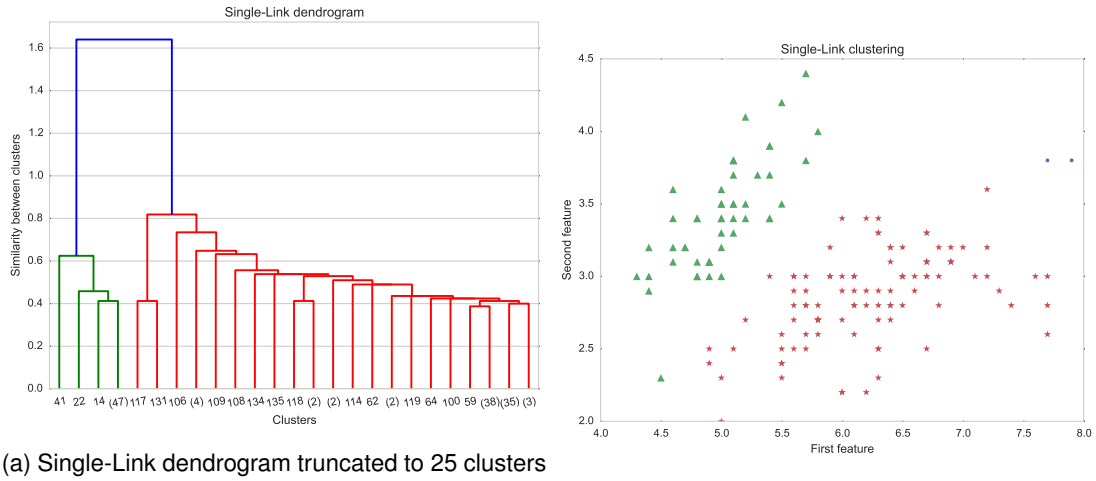
The true advantage of using an MST based approach comes when the number of edges (similarities) m of the MST is less than $\frac{n(n-1)}{2}$, where n is the number of nodes (patterns) [23]. This is because SLINK works over a inter pattern similarity matrix, meaning that the similarity between every pair of patterns must be explicitly represented. The minimum number of similarities is $\frac{n(n-1)}{2}$, which is equivalent to the upper or lower half triangular matrices of the similarity matrix. The MST, on the other hand, works over a graph that may or may not have edges between every pair of nodes. Fast MST algorithms have a time complexity of $O(mlogn)$, which is an improvement over $O(n^2)$ when $m << \frac{n(n-1)}{2}$.



(a) Example of a graph. (b) MST of the graph to the left.

Figure 1.3: The above figures show an example of a graph (left) and its corresponding Minimum Spanning Tree (right). The circles are vertices and the edges are the lines linking the vertices.

An example of a Single-Link dendrogram and resulting cluster can be observed in Fig. 1.4. The dendrogram in Fig. 1.4a has been truncated to 25 clusters in the bottom level for the sake of readability. The clustering presented on Fig. 1.4b is the result of cutting the dendrogram such that only 3 clusters exist (the number of classes). The accuracy, as measured by the CI, is of 58%.



(a) Single-Link dendrogram truncated to 25 clusters in the bottom level.

(b) Clustering with 3 clusters.

Figure 1.4: The above plots show the dendrogram and a possible clustering taken from a Single-Link run over the Iris data set. Fig. 1.4b was obtained by performing a cut on a level that would yield a partition of 3 clusters.

1.6 Ensemble Clustering

The underlying idea behind ensemble clustering is to take a collection of partitions, a *clustering ensemble*, and combine it into a single partition. There are several motivations for ensemble clustering. Data from real world problems appear in different configurations regarding shape, cardinality, dimensionality, sparsity, etc. Different clustering algorithms are appropriate for different data configurations, e.g. K-Means tends to group patterns in hyperspheres [3] 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 its configuration, a single clustering algorithm might perform well for some part of the data while other performs better for some other part. Since different algorithms can be used to produce the partitions in the ensemble, one can use a mix of algorithms to address different properties of the data such that the combination is more **robust** to noise and outliers [24] and the final clustering has a **better quality** [7]. Ensemble clustering can also be particularly useful in situations where one does not have direct access to all the features of a given data set but can have access to partitions from different subsets and later combining with an ensemble algorithm. Furthermore, the generation of the clustering ensemble can be **parallelized and distributed** since each partition is independent from every other partition.

A clustering ensemble, according to [25], can be produced from (1) different data representations, e.g. choice of preprocessing, feature selection and extraction, sampling; or (2) different partitions of the data, e.g. output of different algorithms, varying the initialization parameters on the same algorithm.

Ensemble clustering algorithms can take three main distinct approaches [7]: based on pair-wise similarities, probabilistic or direct. EAC [25] and CSPA [26] are examples of pair-wise similarity based approach, where the algorithms use a co-associations matrix. The MMCE [24] and BCE [27] are examples of a probabilistic approach. This approach will be further clarified when the EAC algorithm is explained. HGPA [26], MCLA [26] and bagging [28] are examples of a direct approach to combining the ensemble clusterings, where the algorithms work directly with the labels without creating a co-association matrix. A detailed and thorough review of the similarity measures that can be used on with clustering ensembles and the state of the art algorithms can be consulted in [7].

1.7 Evidence Accumulation Clustering

1.7.1 Overview

The goal of EAC is to find an optimal partition P^* containing k^* clusters, from the clustering ensemble $\mathbb{P}$. The optimal partition should have the following properties [25]:

- **Consistency** with the clustering ensemble;
- **Robustness** to small variations in the ensemble; and,
- **Goodness** of fit with ground truth information, when available.

Ground truth is the true labels of each sample of the dataset, when such exists, and is used for validation purposes. Since EAC is an unsupervised method, this typically will not be available. EAC makes no assumption on the number of clusters in each data partition. Its approach is divided in 3 steps:

1. **Production** of a clustering ensemble $\mathbb{P}$ (the evidence);
2. **Combination** of the ensemble into a co-association matrix;
3. **Recovery** of the natural clusters of the data.

In the first step, a clustering ensemble is produced. Within the context of EAC, it is of interest to have variety in the ensemble with the intention to better capture the underlying structure of the data. One such parameter to measure that variety is the number of clusters in the partitions of the ensemble. Typically, the number of clusters in each partition is drawn from an interval $[K_{min}, K_{max}]$ with uniform probability. This influences other properties of other parts of the algorithm such as the sparsity of the co-association matrix as will become clearer in future chapters. Reviewing the literature [29, 25, 30, 31], it is clear the ensemble is usually produced by random initialization of K-Means (specifying only the number of centroids within the above interval). Still, other clustering algorithms have been used for the production of the ensemble [32] such as Single-Link, Average-Link and CLARANS.

The ensemble of partitions is combined in the second step, where a non-linear transformation turns the ensemble into a co-association matrix [25], i.e. a matrix $\mathcal{C}$ where each of its elements n_{ij} is the association value between the pattern 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, i.e. the number of co-occurrences in the same cluster. The rationale is that pairs that are frequently clustered together are more likely to be representative of a true link between the patterns [29], revealing the underlying structure of the data. In other words, a high association n_{ij} means it is more likely that patterns i and j belong to the same class. The construction of the co-association matrix is at the very core of this method.

The co-association matrix itself is not the output of EAC. Instead, it is used as input to other methods to obtain the final partition. The co-association between any two patterns can be interpreted as a similarity measure. Thus, since this matrix is a similarity matrix it's appropriate to use algorithms that take this type of matrices as input, e.g. K-Medoids or hierarchical algorithms such as Single-Link or Average-Link, to name a few. Typically, algorithms use a distance as the dissimilarity, which means that they minimize the distance 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 $max\{\mathcal{C}\} - n_{ij}$.

Although any algorithm can be used, the final clustering is usually done using SL or AL. Each of this algorithms will take as input the transformed co-association matrix as the dissimilarity matrix. Furthermore, not knowing the "natural" number of clusters one can use the lifetime criteria, i.e. the number of

clusters k should be such that it maximizes the cost of cutting the dendrogram from $k-1$ clusters to k. Further details on the lifetime strategy for picking the number of clusters falls outside the scope of this work and are presented in [25].

Related work to EAC has been developed. The Weighted EAC (WEAC) algorithm [32] and a study on the sparsity of the co-association matrix [33] should be mentioned. The latter is discussed in more depth in chapter **??**. The former introduces the novelty of having weights associated to each partition such that good quality partitions are more relevant than their counterparts. These weights are based on internal validity measures. Weighing the partitions in terms of quality has shown to improve the original algorithm, accuracy wise.

1.7.2 Examples of applications

EAC has been used with success in several areas. Some of its applications are:

- in the field of bioinformatics it was used for the automatic identification of chronic lymphocyt leukemia [34];
- also in bioinformatics it was used for the unsupervised analysis of ECG-based biometric database to highlight natural groups and gain further insight [31];
- in computer vision it was used as a solution to the problem of clustering of contour images (from hardware tools) [30].

Chapter 2

Results and Discussion

The present chapter is dedicated to the results relevant to the work produced and their associated interpretation and subsequent discussion.

2.1 Experimental environment

All experiments were carried out in one (or more) of three distinct machines what will be referred to as **Alpha**, **Bravo** and **Charlie**. Their CPU and GPU hardware configurations are described in Tables 2.1, 2.2 and 2.3, respectively. Besides, Charlie has a *Seagate ST2000DM001* 7200 RPM spinning disk and a *Samsung 840 EVO* Solid State Drive, informations relevant for the third phase of EAC.

Software wise, all machines are running Linux based operating systems. Alpha and Bravo are using the Ubuntu 14.04 and 12.04, respectively, with a graphical interface. Whether the machine is running a graphical interface or not is important because, in the case that it only has one GPU (as is the case with all the machines here presented) the available memory for computation is less than total and there is a limit to how long a CUDA kernel can be executed. Charlie is running Fedora 21 without a user interface.

Table 2.1: **Alpha** machine specifications.

	CPU	**GPU**
# Devices	1	1
Manufacturer	Intel	NVIDIA
Model	i3-2310M	GT 520M
Launch date	Q1'11	Q1'2011
Architecture	Sandy Bridge	Fermi
# Cores	2	48
Clock frequency [Mhz]	2100	1480
L1 Cache	64KB IC [a] + 64KB DC [b]	16/48 KB/SM [c]
L2 Cache	512KB	n/a
L3 Cache	3 MB	n/a
Memory [GB]	4	1
Max. memory bandwidth [Gbps]	21.3	12.8

[a] Instruction Cache (IC)
[b] Data Cache (IC)
[c] Each Streaming Multiprocessor has 64 KB of on-chip memory that can be configured as either 16KB of L1 cache and 48 KB of shared memory, or vice versa.

Table 2.2: **Bravo** machine specifications.

	CPU	**GPU**
# Devices	1	1
Manufacturer	Intel	NVIDIA
Model	i7 4770K	K40c
Launch date	Q2'13	Q4'13
Architecture	Haswell	Kepler
# Cores	4	2880
Clock frequency [Mhz]	3500	745
L1 Cache	128 KB IC [a] + 128 KB DC [b]	16/48 KB/SM [c] + 48KB DC [d]
L2 Cache	1 MB	1.5 MB
L3 Cache	8 MB	n/a
Memory [GB]	32	12
Max. memory bandwidth [Gbps]	25,6	288

[a] Instruction Cache (IC)
[b] Data Cache (IC)
[c] Each Streaming Multiprocessor has 64 KB of on-chip memory that can be configured as either 16KB of L1 cache and 48 KB of shared memory, or vice versa.
[d] The Kepler architecture has an extra read-only 48KB of Data Cache at the same level of the L1 cache.

Table 2.3: **Charlie** machine specifications.

	CPU	**GPU**
# Devices	1	1
Manufacturer	Intel	NVIDIA
Model	i7-4930K	Quadro K600
Launch date	Q3'13	Q1'2013
Architecture	Ivy Bridge	
# Cores	6	192
Clock frequency [Mhz]	3400	876
L1 Cache	192 KB IC [a] + 192 KB DC [b]	16/48 KB/SM [c] + 48KB DC [d]
L2 Cache	1,5 MB	1.5 MB
L3 Cache	12 MB	n/a
Memory [GB]	32	1
Max. memory bandwidth [Gbps]	59,6	28,5

[a] Instruction Cache (IC)
[b] Data Cache (IC)
[c] Each Streaming Multiprocessor has 64 KB of on-chip memory that can be configured as either 16KB of L1 cache and 48 KB of shared memory, or vice versa.
[d] The Kepler architecture has an extra read-only 48KB of Data Cache at the same level of the L1 cache.

2.2 Parallel K-Means

To test the time efficiency

2.3 GPU MST

To test the performance of the GPU MST algorithm, several graphs were used. Most of the graphs are United Stated road network graphs taken from the 9th DIMACS Implementation Challenge [1]. Furthermore, graphs taken from co-association matrix of the second step of EAC were used. This is important because, as will become clear, the graphs within the EAC paradigm have different characteristics. All the tests were performed on machine Bravo. The average speed-up obtained by using the GPU version over the sequential one is presented in Table 2.4. Characteristics of the different graphs are also shown so as to illustrate what variables influence the speed-up obtained. It should be noted that a speed-up below 0 is actually a slow-down and its absolute value corresponds to the speed-up of the sequential version relative its GPU counterpart.

[1] http://www.dis.uniroma1.it/ challenge9/

Table 2.4: Average speed-up of the GPU MST algorithm for different data sets, sorted by number of edges.

Data set	No. vertices	No. edges	Speed-up [a]	No. edges / vertex	Memory [MBytes]
NY	264347	730100	-1.293761	2.761900	7.587030
BAY	321271	794830	-1.254579	2.474017	8.515170
COL	435667	1042400	-1.004995	2.392653	11.276800
FLA	1070377	2687902	1.389240	2.511173	28.673400
NW	1207946	2820774	1.451060	2.335182	30.736700
NE	1524454	3868020	1.559920	2.537315	41.141300
CAL	1890816	4630444	1.584020	2.448913	49.753300
LKS	2758120	6794808	1.699390	2.463565	72.883100
E	3598624	8708058	1.803500	2.419830	93.892500
W	6262105	15000000	1.935430	2.395361	163.127052
Coassoc 50k [b]	50000	30296070	-4.967957	605.921400	231.522141
CTR	14000000	34000000	2.088050	2.428571	365.819000

[a] Average speed-up from 10 rounds of executing the algorithm on each graph.
[b] Co-association matrix of a 100 partition ensemble produced from a mixture of 6 Gaussians with $50\,000$ patterns, using the rule sk=sqrt_2 th=30%.

Although all the graphs presented in Table 2.4 occupy significantly less memory than the available in the used machine, the processing of bigger graphs is not possible. The reason for this is that between in each iteration two graphs have to be held in memory: the initial and the contracted. Moreover, the space occupied by the contracted graph will depend on the characteristics of the graph.

The results clearly show that it is possible to obtain speed-ups for computing a MST. This speed-up seems to increase with the size of the graph, with the notable exception of the graph from the EAC context. A note should be made here to bring to attention the contrast between these results and those presented by Sousa et al. [35]. The speed-ups observed here are less than those reported by Sousa et al. [35]. This is believed to be related with the technology stack used and this topic has been discussed in more depth in chapter **??**. To understand how different parameters affect the speed-up of the algorithm, Table 2.5 presents the correlation matrix of these variables.

Table 2.5: Cross-correlation between several characteristics of the graphs and the average speed-up.

	No. vertices	No. edges	Average speed-up	No. edges / vertex
No. vertices	1.000000	0.670382	0.692747	-0.217886
No. edges	0.670382	1.000000	0.081862	0.578078
Average speed-up	0.692747	0.081862	1.000000	-0.653624
No. edges / vertex	-0.217886	0.578078	-0.653624	1.000000

The row corresponding to the average speed-up is of special relevance. One can observe that the parameters most correlated with the speed-up are the number of vertices and the number of edges per vertex (EPV). The correlation matrix suggests that as one increases the number of vertices, the

speed-up will also increase. In fact, if no graphs from the EAC context were present in the results, the same would apply to the number of edges, since the EPV would very similar. The reason for this is that speed-ups from parallelism are more salient when applied to big data sets, so that the speed-up of the computation itself outweighs the overhead associated with communication between host and device. The EPV is the other parameter that shows has highest (inverse) correlation with the speed-up. This suggests that the relationship between the number of edges and the number of vertices in the graph actually plays a big role in deciding if there will be a speed-up.

The underlying reason for the poor performance of graphs with high EPV ratio is believed to be that, since the parallel computation is anchored to vertices, the workload per vertex is higher than if the graph had a low ratio. Accordingly, the workload per vertex is higher from the beginning and can increase significantly as the algorithm progresses. Besides, the workload can become highly unbalanced with some threads having to process hundreds of thousands of edges while others only a few thousands, which translates threads not doing any computation when waiting for the others.

The original source [35] of the algorithm doesn't address graphs with a EPV as high as presented here. In that sense, the results here complement those of the original source and suggest an increase in EPC translates in the decrease in speed-up. Still, more in-depth studies should be made.

Within EAC paradigm, this algorithm is of little contribution. The most obvious reason is that the EAC method would actually be slower if this algorithm was used. Still, even considering that speed-ups like those reported in literature were possible for EAC co-association graphs, the algorithm requires a double redundancy of edges (which effectively doubles the necessary memory to hold a graph) and at any iteration the device must be able to hold two distinct graphs (the initial and the contracted). For these reasons, the device memory (which typically is smaller than the host memory) would confine the EAC method to small input data sets.

2.4 Building the co-association matrix with different sparse formats

The purpose of this section is presenting brief results concerning the time that took to build a co-association matrix for different types of matrices. The ensemble from which the co-association matrices are built has 100 partitions and was produced from a mixture of 6 Gaussians with 5000 patterns. Only the upper triangular (condensed) matrix was built. The types of matrices under test are: fully allocated (a "normal" matrix), LIL, DOK, CSR, an optimized fully allocated and the proposed EAC CSR. The SciPy's LIL, DOK and CSR implementations were used.

The time that took to update the first partition and the total time were recorded for the different types of matrix. The results can are presented in Table 2.6 and also in Fig. 2.1. For the CSR format only the first partition was updated, since it took a very long time to update just the first partition. A rough estimate for the time it would take to update the whole matrix is around 15h, 100 times the time it took to update the first partition. Observing the other timings, and for the exception of the EAC CSR matrix,

this estimate should not be too far off. The reason that the first partition update of the EAC CSR matrix was so much faster is that it only requires a simple copy of the partition to the data structure.

It is clear from the results that the optimized versions are much faster than any of the others. These results focus on providing a justification for the design and implementation of a novel method of building the co-association matrix: a fully allocated matrix consumes too much memory but available sparse implementations are too slow. For this purpose a small data set as the one used suffices to demonstrate this point. The difference between the two optimized versions will become clearer on future sections, where a more thorough study covering a wider spectrum of data sets is presented.

Table 2.6: Times for computing the condensed co-association matrix using different matrix strategies.

Matrix type (condensed)	Time 1st partition [s]	Time ensemble [s]
Optimzed Fully allocated	0.00170	0.139
EAC CSR	0.00481	1.470
Fully allocated	0.85500	96.000
LIL	5.39000	614.000
DOK	12.50000	1535.000
CSR	548.00000	-

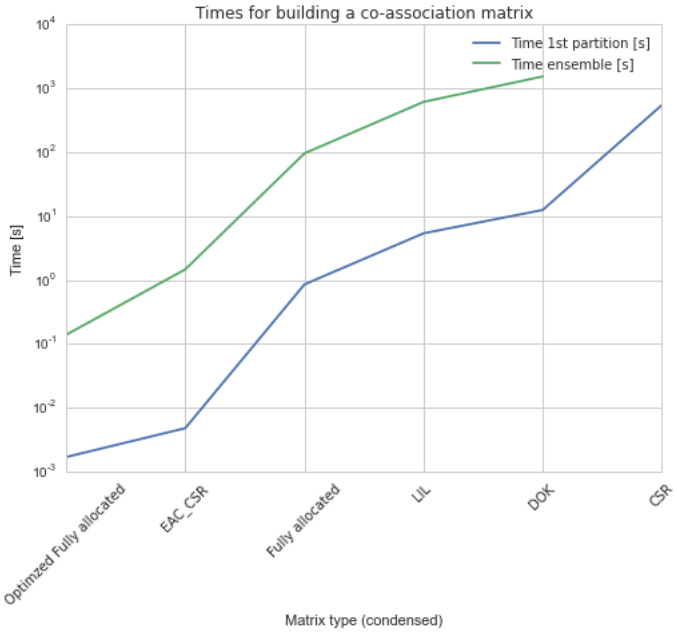


Figure 2.1: sd

2.5 EAC Validation

The present section aims to provide results showing that the proposed methods do not alter the overall quality of the results. With this in mind, the results of original version of EAC, implemented in Matlab, are compared with those of the proposed solution. Several small data sets, chosen from the data sets used by Lourenço et al. [33], were processed by the two versions of EAC. All these data sets were

taken from the UCI Machine Learning Repository. Furthermore, since the generation of the ensemble is probabilistic and can change the results between runs, the proposed version is processed with the ensembles created by the original version as well. This guarantees that the combination and recovery phases of EAC, which are deterministic when using SL, are equivalent to the original. All data in this section refers to processing done in machine Alpha.

Table 2.7 presents the difference between the accuracies of the two versions. Analyzing these results, it is apparent that the difference is minimal. It should be noted at this point that the original implementation always maps the dissimilarities of the co-association matrix to the range $[0, 1]$. This forces the co-association matrix to have a floating point data type. However, since the number of partitions used is usually less than 255, the proposed version uses unsigned integers of 1 byte to reduce the used memory considerably. The differences in accuracy are thought to come from rounding differences of the two frameworks used and from this difference in data type.

Table 2.7: Difference between accuracies from the two implementations of EAC, using the same ensemble. Accuracy was measured using the H-index.

Data set	Accuracy	Accuracy (lifetime)
breast_cancer	4.948755e-06	2.825769e-06
ionosphere	1.652422e-06	1.452991e-06
iris	3.333333e-06	3.333333e-06
isolet	1.038861e-07	4.084904e-07
optdigits	3.795449e-06	1.480513e-06
pima	3.333333e-06	3.333333e-06
pima_norm	4.166667e-07	4.166667e-07
wine_norm	1.123596e-07	1.910112e-06

Table 2.8 presents the speed-up of the proposed version over the original one. It is clear that speed-up is obtained in all phases of EAC, often by an order of magnitude. This result, combined with the demonstration that the differences in accuracy are negligible, show that the proposed algorithm performs well in small data sets.

Table 2.8: Speed-ups obtained in the different phases of EAC, with independent production of ensembles.

Data set	No. patterns	No. features	No. classes	Production	Combination	Recovery
breast_cancer	683	10	2	50.43974	7.544247	15.83316
ionosphere	351	34	2	21.86286	11.30883	19.97219
iris	150	4	3	19.76525	14.49562	28.50479
isolet	7797	617	26	7.010007	6.183124	206.2837
optdigits	3823	64	10	17.30209	10.2096	53.02636
pima	768	8	2	50.65624	141.4828	13.93502
pima_norm	768	8	2	54.25415	132.8632	14.355
wine_norm	178	4	3	22.92404	14.56994	25.27709

2.6 EAC

This section will present thorough results concerning several characteristics related to the EAC method.

2.7 Quantum K-Means

The algorithm implemented and tested is a variant of the one described in [**?**]. 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 [36]. This decision was based on the findings of [37], 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.

2.7.1 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 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 2.9: 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

Table 2.10: 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. 2.2 and 2.3) 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.

Analysing the evolution of the DB index of the best solution over the generations (Fig. 2.4 and 2.5)

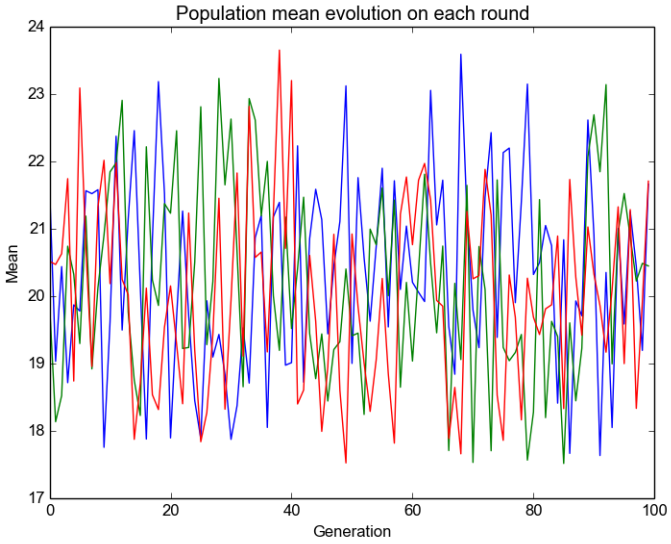


Figure 2.2: DB index mean of the population in T2. Only 4 rounds represented.

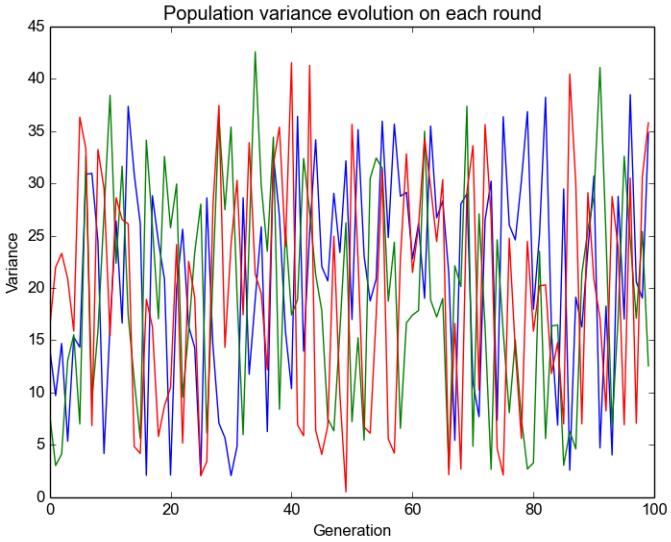


Figure 2.3: DB index variance of the population in T2. Only 4 rounds represented.

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 2.11.

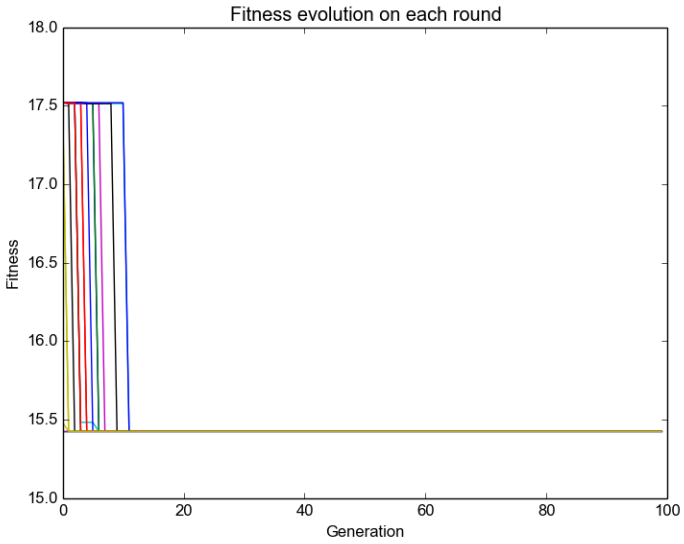


Figure 2.4: DB index of best solution in T2.

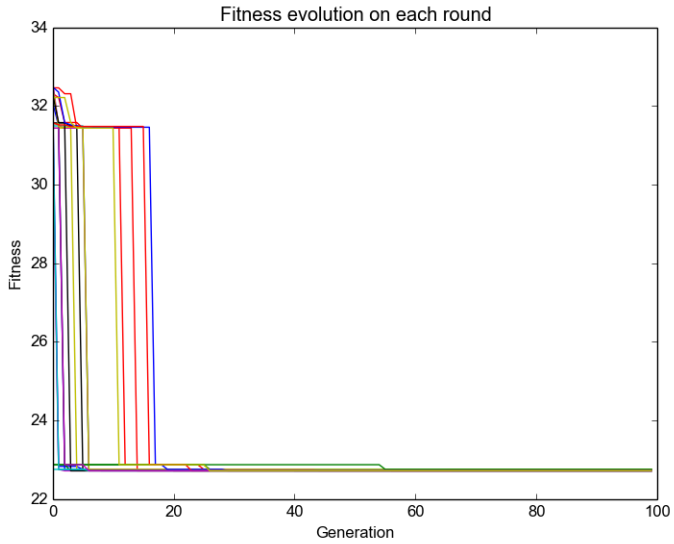


Figure 2.5: DB index of best solution in T3.

2.7.2 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 does not require its input partitions to be accurate, the quality of the solutions, measured with the Davies-Bouldin index, from QK-Means does not 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.

Table 2.11: The values represent generations.

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

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.

2.8 Horn and Gottlieb's algorithm

All data in this section refers to processing done in machine Alpha.

2.8.1 Performance

Table 2.12: Time of computation of Horn and Gottlieb [1] algorithm for a mixture of 4 Gaussians of different cardinality and dimensionality.

Cardinality	Dimensionality	Time [s]
10	2	0.035382
10	3	0.411391
10	4	0.385114
10	5	0.429747
100	2	2.954650
100	3	3.322593
100	4	3.743720
100	5	4.143823
1000	2	52.840666
1000	3	60.293262
1000	4	68.225671
1000	5	81.523212
10000	2	3009.678259
10000	3	3418.342830
10000	4	3956.289064
10000	5	4918.185844

2.8.2 Accuracy

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

2.8.3 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. 2.6.

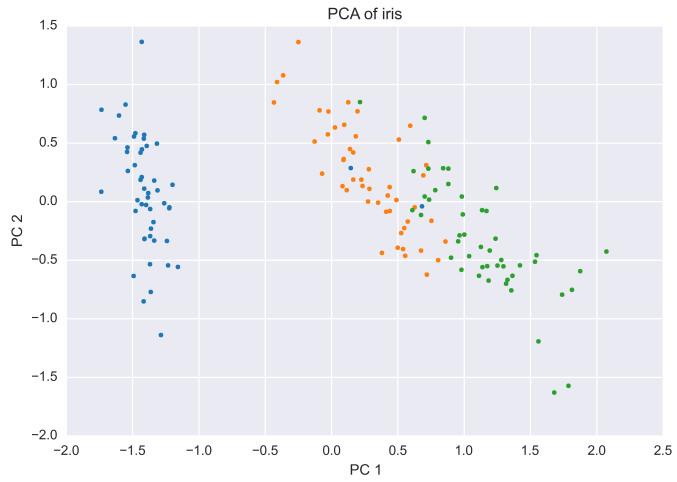


Figure 2.6: Plot of the two first principal components (PC).

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. 2.7 and have an accuracy of 86% computed with consistency index.

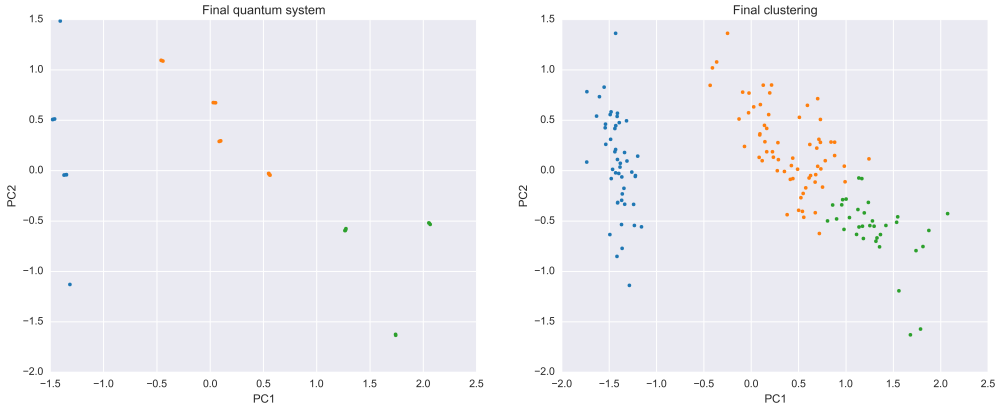


Figure 2.7: Plots of the converged data data points and final clustering for 2 PC.

For the sake of completeness, Fig. 2.8 shows the clustering over all PCs. This solution has an accuracy of 82.67% computed with consistency index.

2.8.4 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. **??**.

Initial work aimed at reproducing results from [2], but lack of detail on the preprocessing used made

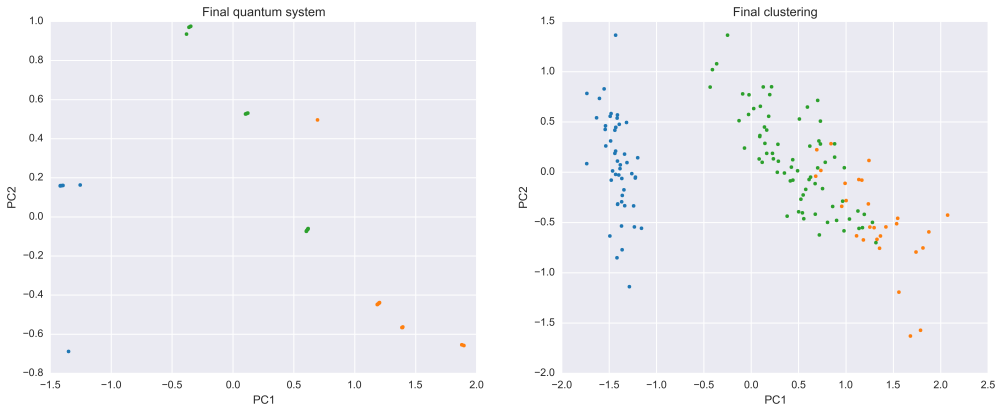


Figure 2.8: Plots of the converged data data points and final clustering for all PC of Iris data.

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 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

Chapter 3

Discussion?

3.1 real num assocs compared to samples per cluster

[33] reports that, on average, the overall contribution of the clustering ensemble (including unbalanced clusters) duplicates the co-associations produced in a single balanced clustering with Kmin clusters. However, the spectrum of datasets evaluated regarding cardinality was smaller than that evaluated in the present work. The results suggest that the contribution of the ensemble is in fact higher.

3.2 Trade-off speed accuracy memory

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.

3.3 Expanding this work to other scalability paradigms

The present work focused on two main approaches for scalability: GPGPU and out-of-core solution. A current trend in computing of big data is cluster computing, which allows for distributed and parallel computing. For the sake of completeness it is interesting to discuss if the ideas related to the former two paradigms are transferable to the later.

The concept of GPGPU is one of parallelization. It is based on the fact that each computing thread in the GPU will execute instructions on a restricted subset of the entire dataset. This idea is easily transferable to cluster computing, as it is a core concept on both paradigms. For this reason, the generation of the ensemble is a step of EAC that can be very easily transfered to a computing cluster.

Bibliography

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

[2] Anil 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.

[3] a. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3): 264–323, 1999. ISSN 03600300. doi: 10.1145/331499.331504.

[4] Edgar Anderson. The species problem in Iris. *Annals of the Missouri Botanical Garden*, pages 457–509, 1936.

[5] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7 (2):179–188, 1936.

[6] Raymond A Jarvis and Edward A Patrick. Clustering using a similarity measure based on shared near neighbors. *Computers, IEEE Transactions on*, 100(11):1025–1034, 1973.

[7] Charu C Aggarwal and Chandan K Reddy. *Data clustering: algorithms and applications*. CRC Press, 2013. ISBN 9781466558229.

[8] Ana Fred. Finding consistent clusters in data partitions. *Multiple classifier systems*, pages 309–318, 2001. URL http://link.springer.com/chapter/10.1007/3-540-48219-9_31.

[9] Marina Meila. Comparing clusterings by the variation of information. *Learning theory and Kernel machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003: proceedings*, page 173, 2003. ISSN 03029743. doi: 10.1007/978-3-540-45167-9_14. URL http://books.google.com/books?hl=en&lr=&id=hk1dqsM0XF4C&oi=fnd&pg=PA173&dq=Comparing+Clusterings+by+the+Variation+of+Information&ots=7rcmrLpFV1&sig=P-AXGQnlenPfAlSb3fdhphYv6dI.

[10] David L Davies and Donald W Bouldin. A cluster separation measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):224–227, 1979.

[11] Daniel Boley. Principal Direction Divisive Partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998. ISSN 1384-5810. doi: 10.1023/A:1009740529316. URL http://dx.doi.org/10.1023/A:1009740529316$\delimiter"026E30F$nhttp://www.springerlink.com/content/w15313n737603612/.

[12] Michael Steinbach, George Karypis, Vipin Kumar, and Others. A comparison of document clustering techniques. *KDD workshop on text mining*, 400(1):525–526, 2000. doi: 10.1.1.125.9225.

[13] Marie Chavent. A monothetic clustering method. *Pattern Recognition Letters*, 19(11):989–996, 1998. ISSN 01678655. doi: 10.1016/S0167-8655(98)00087-7.

[14] James C. Bezdek, Robert Ehrlich, and William Full. FCM: The fuzzy ¡i¿c¡/i¿-means clustering algorithm. *Computers & Geosciences*, 10(2–3):191–203, 1984. ISSN 0098-3004. doi: 10.1016/0098-3004(84)90020-7. URL http://www.sciencedirect.com/science/article/pii/0098300484900207.

[15] U Kang, Charalampos E Tsourakakis, and Christos Faloutsos. PEGASUS : Mining Peta-Scale Graphs. *Knowledge and Information Systems*, 27(2):303–325, 2011.

[16] J B MacQueen. Some Methods for classification and Analysis of Multivariate Observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press*, volume 1, pages 281–297, 1967.

[17] D. Arthur, D. Arthur, S. Vassilvitskii, and S. Vassilvitskii. k-means++: The advantages of careful seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 8: 1027–1035, 2007. doi: 10.1145/1283383.1283494. URL http://portal.acm.org/citation.cfm?id=1283494.

[18] Malay K Pakhira. A Modified k -means Algorithm to Avoid Empty Clusters. *International Journal of Recent Trends in Enineering*, 1(1), 2009.

[19] Peter H A Sneath and Robert R Sokal. Numerical taxonomy. *Nature*, 193(4818):855–860, 1962.

[20] R. Sibson. SLINK: an optimally efficient algorithm for the single-link cluster method, 1973. ISSN 1460-2067. URL http://comjnl.oxfordjournals.org/content/16/1/30.short.

[21] J. C. Gower and G. J. S. Ross. Minimum Spanning Trees and Single Linkage Cluster Analysis. *Journal of the Royal Statistical Society*, 18(1):54–64, 1969.

[22] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. (1973):29, 2011. URL http://arxiv.org/abs/1109.2378.

[23] J-L Starck and Fionn Murtagh. *Astronomical image and data analysis*. Springer Science & Business Media, 2007.

[24] Alexander Topchy, Anil K Jain, and William Punch. A mixture model for clustering ensembles. In *Society for Industrial and Applied Mathematics. Proceedings of the SIAM International Conference on Data Mining*, page 379. Society for Industrial and Applied Mathematics, 2004.

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

[26] Alexander Strehl and Joydeep Ghosh. Cluster Ensembles – A Knowledge Reuse Framework for. *Journal of Machine Learning Research*, 3:583–617, 2002. ISSN 1532-4435. doi: 10.1162/153244303321897735.

[27] Hongjun Wang, Hanhuai Shan, and Arindam Banerjee. Bayesian cluster ensembles. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 4(1):54–70, 2011.

[28] Sandrine Dudoit and Jane Fridlyand. Bagging to improve the accuracy of a clustering procedure. *Bioinformatics*, 19(9):1090–1099, 2003. ISSN 13674803. doi: 10.1093/bioinformatics/btg038.

[29] Ana N L Fred and Anil K Jain. Data clustering using evidence accumulation. *Object recognition supported by user interaction for service robots*, 4, 2002. ISSN 1051-4651. doi: 10.1109/ICPR.2002.1047450.

[30] André Lourenço and Ana 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.

[31] André Lourenço, Carlos Carreiras, Samuel Rota Bulò, and Ana Fred. ECG ANALYSIS USING CONSENSUS CLUSTERING. pages 511–515, 2009. URL `Lourenco2009`.

[32] F.J. Duarte, a.L.N. Fred, a. Lourenco, and M.F. Rodrigues. Weighting Cluster Ensembles in Evidence Accumulation Clustering. *2005 Portuguese Conference on Artificial Intelligence*, 00:159–167, 2005. doi: 10.1109/EPIA.2005.341287.

[33] André Lourenço, Ana L N Fred, and Anil 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.

[34] You Wen Qian, William Cukierski, Mona Osman, and Lauri 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.

[35] Cristiano da Silva Sousa, Artur Mariano, and Alberto Proença. A Generic and Highly Efficient Parallel Variant of Boruvka 's Algorithm. 2015. URL `https://github.com/Beatgodes/BoruvkaUMinho`.

[36] Nathan Wiebe, Ashish Kapoor, and Krysta Svore. Quantum Algorithms for Nearest-Neighbor Methods for Supervised and Unsupervised Learning. page 31, 2014. URL `http://arxiv.org/abs/1401.2142`.

[37] Wenjie Liu, Hanwu Chen, Qiaoqiao Yan, Zhihao Liu, Juan Xu, and Yu 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.