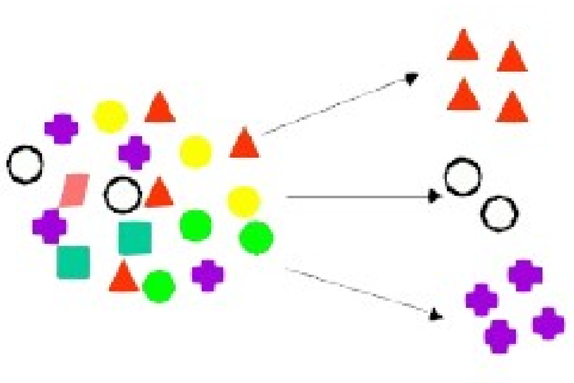


Efficient Evidence Accumulation Clustering for large datasets/big data

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

Supervisors: Dra. Ana Fred and Dra. Helena Aidos

Examination Committee

Chairperson:	
Supervisor:	Dra. Ana Luísa Nobre Fred
Co-supervisor:	Dra. Helena Aidos Lopes
Member of the Committee:	

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

Avanços na tecnologia permitem a recolha e armazenamento de quantidades e variedades de dados sem precedente. A maior parte destes dados são armazenados eletrónicamente e existe interesse em realizar análise automática dos mesmos. As técnicas de *clustering* estão entre as mais populares para essa tarefa porque não assumem nada sobre a estrutura dos dados *a priori*. Dezenas de técnicas existem, mas, típicamente, uma só técnica não tem um bom desempenho em todos os conjuntos de dados devido às especifidades de cada um. Técnicas de *ensemble clustering* tentam responder a esse desafio ao combinar outros algoritmos. Esta dissertação foca-se numa em particular, o *Evidence Accumulation Clustering* (EAC). O EAC é uma algorithm robusto que tem demonstrado bons desempenhos na literatura numa variedade de conjuntos de dados. No entanto, esta robustez vem com um maior custo computacional associado. A sua aplicação não só é mais lenta como está restrita a conjuntos de dados pequenos. Assim, o objetivo desta dissertação é escalar o EAC, possibilitando a sua a aplicação a conjuntos de dados grandes (mais que $500\,000$ amostras), com tecnologia disponivel numa típica estação de trabalho. Com isto em mente, várias abordagens foram exploradas: acelerar processamento com outros algoritmos (*quantum clustering*), através de processamento paralelo (com GPU), escalar para maiores conjuntos de dados com algoritmos de memória externa (disco rígido) e explorando a natureza esparsa do EAC. Uma contribuição de relevo é um método novo para construir uma matriz de esparsa específico ao EAC. Os resultados mostram que a solução proposta é, efetivamente, aplicável a conjuntos de dados grandes e é significativamente mais rápida que a original para conjuntos pequenos.

Palavras-chave: clustering, EAC, quantum clustering, K-Means, Single-Link, matriz esparsa, GPGPU

Abstract

Advances in technology allow for the collection and storage of unprecedented amount and variety of data. Most of this data is stored electronically and there is an interest in automated analysis for generation of knowledge and new insights. Since, often, the structure of the data is unknown, clustering techniques become particularly interesting for knowledge discovery and data mining, since they makes as few assumptions on the data as possible. A vast body of work on these algorithms exist, yet, typically, no single algorithm is able to respond to the specificities of all data. Ensemble clustering algorithm address this problem by combining other algorithms. Evidence Accumulation Clustering (EAC) is a robust ensemble algorithm that has shown good results and is the focus of this dissertation. However, this robustness comes with higher computational cost. Its application is slower and restricted to smaller data sets. Thus, the objective of this dissertation is to scale EAC, allowing its applicability to big data sets (more than $500\,000$ patterns), with technology available at a typical workstation. Accordingly, several approaches were explored: speed-up with other algorithms (*quantum clustering*) or parallel computing (with GPU) and reducing space complexity by using external memory (hard drive) algorithms and exploiting the sparse nature of EAC. A relevant contribution is a novel method to build a sparse matrix specialized in EAC. Results show that the proposed solution is applicable to large data sets and is significantly faster than the original for smaller data sets.

Key words: clustering, EAC, quantum clustering, K-Means, Single-Link, sparse matrix, GPGPU

Contents

Acknowledgments v

Resumo vii

Abstract ix

List of Tables xiii

List of Figures xv

Glossary xvii

1 Clustering: basic concepts, definitions and algorithms 1

1.1 The problem of clustering 1

1.2 Definitions and Notation 2

1.3 Characteristics of clustering techniques 4

1.4 K-Means 5

1.5 Single-Link 7

1.6 Ensemble Clustering 9

1.7 Evidence Accumulation Clustering 9

1.7.1 Overview 9

1.7.2 Examples of applications 11

2 Results and Discussion 12

2.1 Quantum Clustering 12

2.1.1 Quantum K-Means 12

2.1.2 Horn and Gottlieb’s algorithm 13

3 Conclusions 15

3.1 Achievements 15

3.2 Future Work 15

Bibliography 19

List of Tables

2.1 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. 13
2.2 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). 13
2.3 Time of computation of Horn and Gottlieb [1] algorithm for a mixture of 4 Gaussians of different cardinality and dimensionality. 14

List of Figures

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. 2

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. 6

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. 8

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. 8

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

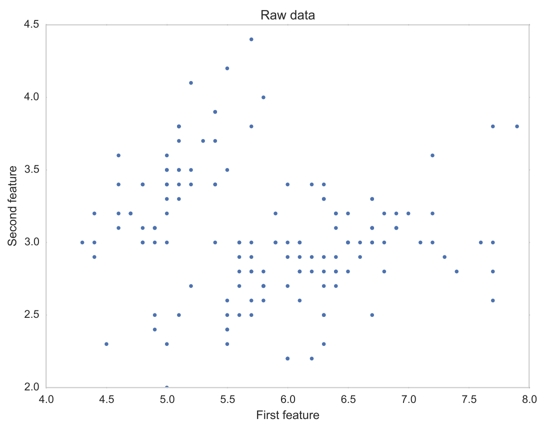
Cluster analysis methods are unsupervised and the backbone of the present work. The goal of data clustering, as defined by [2], is the discovery of the *natural grouping(s)* of a set of patterns, points or objects. In other words, the goal of data clustering is to discover structure on data. The methodology 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.

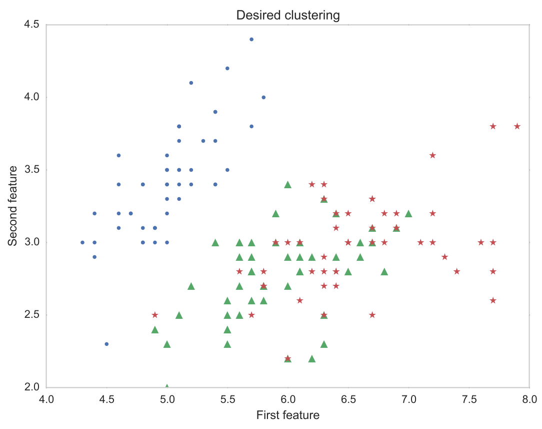
- **Pattern similarity** refers to the definition of a measure for computing the similarity between two patterns.

- **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:

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

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:

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

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.

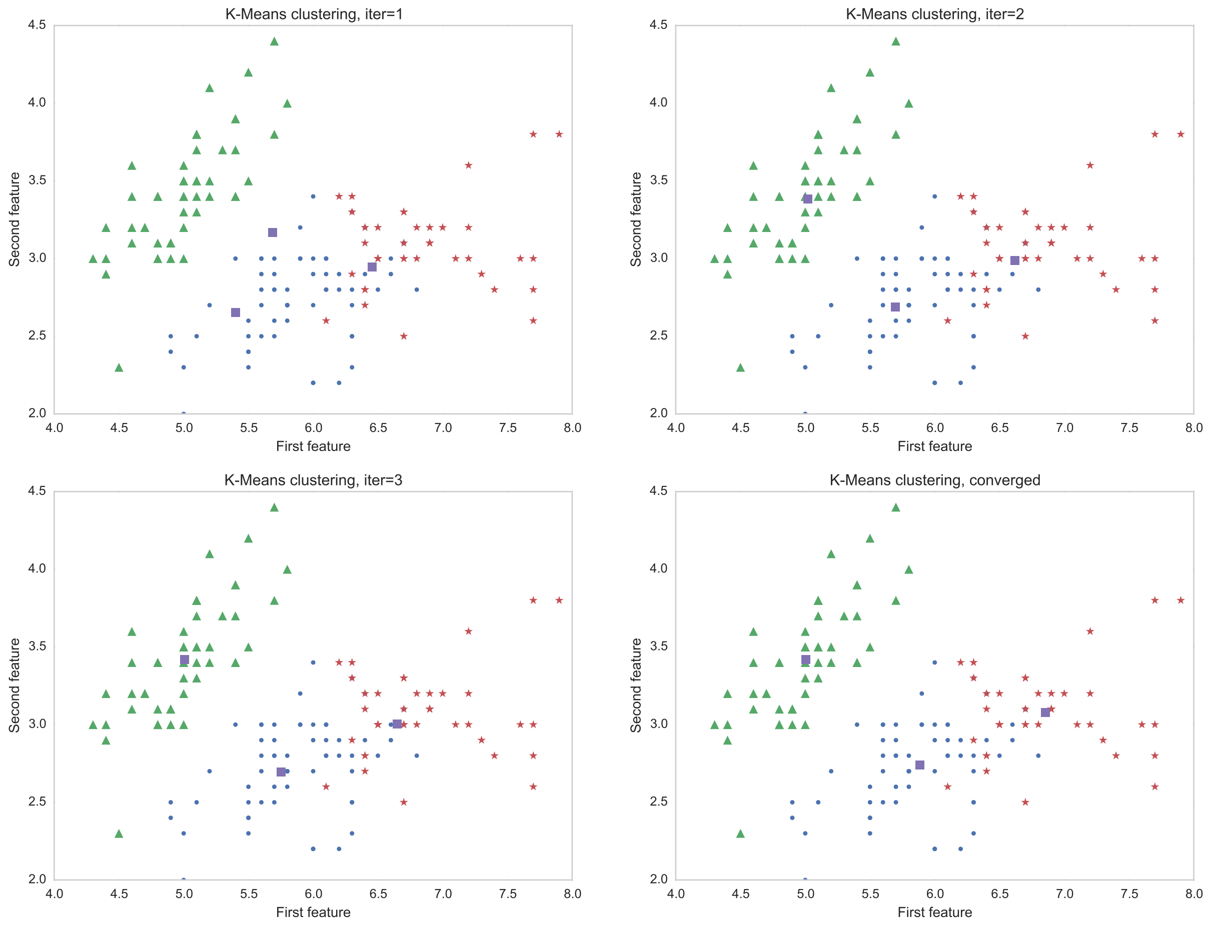


Figure 1.2: The output labels of the K-Means algorithm with the number of clusters (input parameter) set to 3. The different plots show the centroids (squares) evolution on each iteration. Between iteration 3 and the converged state 2 more iterations were executed.

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.

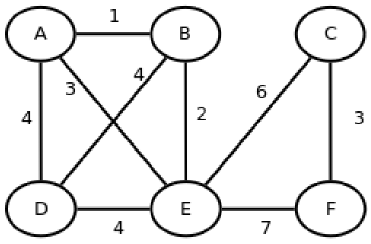
The algorithm stops when $n-1$ merges have been performed, which is when all patterns have been connected in the same cluster. Just like in the K-Means algorithm, different similarity measures can be used for the distances.

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].

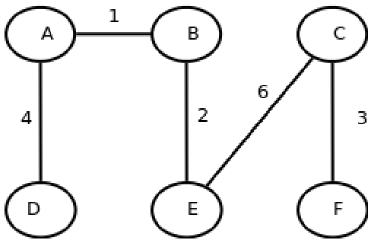
The total time complexity of a naive implementation is $O(n^3)$ since it performs a $O(n^2)$ search in step two and it does it $n-1$ times. Over time, more efficient implementations have been proposed, such as SLINK [20]. SLINK needs no working copy of $O(n^2)$ the pair-wise similarity matrix (if the 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 $m << \frac{n(n-1)}{2}$.



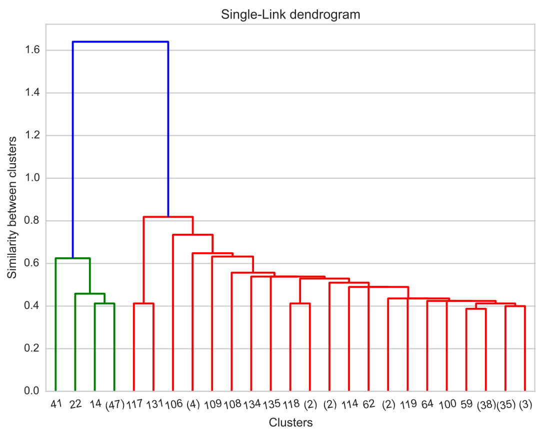
(a) Example of a graph.



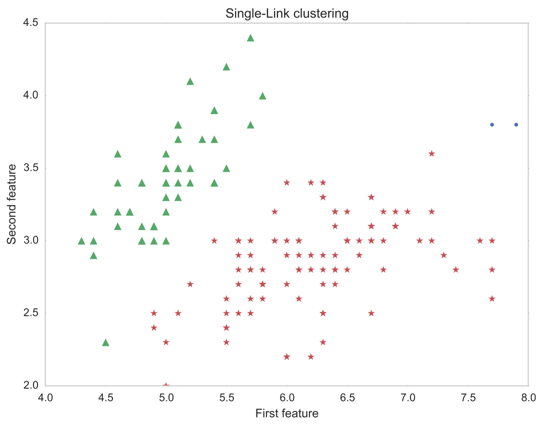
(b) MST of the graph to the left.

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 Quantum Clustering

This section presents performance and accuracy results of Quantum K-Means and Horn and Gottlieb's algorithm. All results were obtained using machine Alpha.

2.1.1 Quantum K-Means

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 tests were performed using 10 oracles, a qubit string length of 8 and 100 generations per round. The classicalK-Means was executed using the *k-means++* centroid initialization method to make for a fairer comparison, 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 performance results for K-Means alone refer to $num.oracles \times num.generations \times factor$ runs, where $factor$ is an adjustable multiplier. Each test had 20 rounds to 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 $factor = 1.10$.

Performance results are presented in Table 2.1. The mean computation time of classical K-Means is an order of magnitude lower than that of QK-Means. However, the solution chosen in classical K-Means was the one with lowest sum of squared euclidean distances of points to their attributed centroid. To analyze the influence of the Davies-Bouldin (DB) index, it was computed on all classical K-Means solutions 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, the computation times become very similar with only a 2% difference between these two variants. Results show that most computational cost (88% on T1) lies on the evaluation of the solutions obtained from each oracle, i.e. computing the DB index. This is a costly but necessary step in this algorithm.

Table 2.1: 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

DB index results are presented in Table 2.2 These results show that the quality of the solutions from K-Means and QK-Means do not differ significantly on these data sets. The results presented before steered the direction of the analysis to a ”fairer” comparison. Yet, the main requirement for the target application of this algorithm is speed. In this regard, it is several orders of magnitude slower than the classical K-Means, since the K-Means performance results refer to many runs. This, allied with the fact that the quality of the solutions does not differ much in the two algorithms and the fact that good quality is not necessary in the target application, makes the use of this algorithm prohibitive in the EAC context.

Table 2.2: 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
T1	QK-Means	15.42531927	32.29577426	19.94704511	21.23544567
	K-Means	15.42531927	25.44913817	16.25013365	1.216919278
T3	QK-Means	22.72836641	65.19984617	36.10699242	78.14043743
	K-Means	22.71934191	46.72231967	26.18440481	22.96730826

2.1.2 Horn and Gottlieb’s algorithm

All data in this section refers to processing done in machine Alpha. For measuring the performance of this algorithm, several mixtures of 4 Gaussians with different cardinality and dimensionality were produced. Table 2.3 presents the execution times for each of these data sets. It can be seen that the execution time of this algorithm is rather high even for small data sets as the ones presented, since this algorithm is being analyzed with the goal of speed optimization. More than 1 hour for a small data set such as $10\,000$ patterns is prohibitive for application in the EAC context.

Table 2.3: 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

In spite of its computational complexity, the accuracy of the algorithm was tested on the Iris and Crab data sets. The Iris data set has 150 patterns, 4 features and 3 classes where 2 are overlapping. The Crab data set has 200 patterns, 5 features and 4 classes where 2 pairs of classes are overlapping. Principal Component Analysis (PCA) was applied to both data sets before clustering. An accuracy as high as 86% was obtained for the Iris data set when using the first and second principal components (PC), and as high as 82% when using all PCs. For the Crab data set an accuracy of 81.5% was obtained for the second and third PCs (PCs chosen to reproduce the results of the original source of the algorithm), 63% when using all PCs. However, when applied to the unprocessed data the accuracy was only 34% .

Chapter 3

Conclusions

Insert your chapter material here...

3.1 Achievements

The major achievements of the present work...

3.2 Future Work

Much effort was put developing and testing co-association matrix building strategies. The schemes presented here provide a solid framework to work with large data sets in this middle step of EAC. As such, interesting directions for this work to take are testing how state of the art algorithms designed for Big Data would complement EAC in the first and last steps.

The programming model used for the GPU was CUDA, used through a Python library called Numba. This library offers an interface to access most of CUDA's capabilities, but not all. One of those capabilities is Dynamic Parallelism. This offers the ability of having a host kernel call on other host kernel without intervention from the host. This translates in the possibility of moving the control logic in the Borůvka variant (and also its the Connected Components Labeling variant) to the device, effectevely removing the memory transfer of values related with the control logic.

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.

A good follow-up of the present work is to study the relationship between several metrics (e.g. sparsity, accuracy, maximum number of co-associations) and the complexity of the dataset. Some metrics for describing the complexity of the dataset exist (Tin Kam Ho) and it would be interesting to profile several datasets of different complexities and structures and search for the former relationship. On a performance perspective it could prove useful to deduce better rules to set the maximum number of associations in the sparse matrix. On a accuracy perspective it would be interesting to see if there are types of datasets that simply are not a good fit for EAC while other are. It would also be interesting to relate complexity with the threshold cut-off.

The WEAC algorithm is focused on improving accuracy. The underlying concept is to measure the quality of the partitions in the ensemble and allow the better ones to be more influential in the co-association matrix. The concept of measuring the quality of the partitions may prove useful for further decreasing the memory complexity with the EAC CSR scheme without compromising too much accuracy. The basic idea is to choose a max_assocs value that will likely be less than the number of associations many patterns will have, which will result in some associations being discarded. The associations that will be kept are the ones from the first partitions that were processed. With this in mind, one could order the partitions by quality and start the processing from those with better quality.

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&amp;lr=&amp;id=hk1dqsM0XF4C&amp;oi=fnd&amp;pg=PA173&amp;dq=Comparing+Clusterings+by+the+Variation+of+Information&amp;ots=7rcmrLpFV1&amp;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.