# Strategies and Algorithms for Clustering Large Datasets: A Review

*Javier Béjar*
*Departament de Llenguatges i Sistemes Informàtics*
*Universitat Politècnica de Catalunya*
*bejar@lsi.upc.edu*

**Abstract**

The exploratory nature of data analysis and data mining makes clustering one of the most usual tasks in these kind of projects. More frequently these projects come from many different application areas like biology, text analysis, signal analysis, etc that involve larger and larger datasets in the number of examples and the number of attributes. Classical methods for clustering data like K-means or hierarchical clustering are beginning to reach its maximum capability to cope with this increase of dataset size. The limitation for these algorithms come either from the need of storing all the data in memory or because of their computational time complexity.

These problems have opened an area for the search of algorithms able to reduce this data overload. Some solutions come from the side of data preprocessing by transforming the data to a lower dimensionality manifold that represents the structure of the data or by summarizing the dataset by obtaining a smaller subset of examples that represent an equivalent information.

A different perspective is to modify the classical clustering algorithms or to derive other ones able to cluster larger datasets. This perspective relies on many different strategies. Techniques such as sampling, on-line processing, summarization, data distribution and efficient datastructures have being applied to the problem of scaling clustering algorithms.

This paper presents a review of different strategies and clustering algorithms that apply these techniques. The aim is to cover the different range of methodologies applied for clustering data and how they can be scaled.

**Keywords:** Unsupervised Learning, Scalable Algorithms, Scalability Strategies

## 1 Introduction

According to a recent poll about the most frequent tasks and methods employed in data mining projects (KDNuggets, 2011), clustering was the third most frequent task. It is usual that these projects involve areas like astronomy, bioinformatics or finance, that generate large quantities of data. Also according to a recurrent poll of KDNuggets the most frequent size of the datasets being processed has shifted from tens of gigabytes in 2011 to terabytes in 2013. It is also common also that, in some of these domains, data is a continuous stream representing and boundless dataset, that is collected and processed in batches to incrementally update or refine a previously built model.

The classical methods for clustering (e.g.: K-means, hierarchical clustering) are not able to cope with this increasing amount of data. The reason is mainly because either the constraint of maintaining all the data in main memory or the temporal complexity of the algorithms. This makes them impractical for the purpose of processing these increasingly larger datasets. This means that the need of scalable clustering methods is a real problem and in consequence some new approaches are being developed.

There are several methodologies that have been used to scale clustering algorithms, some inspired in methodologies successfully used for supervised machine learning, other specific for this unsupervised task. For instance, some of these techniques use different kinds of sampling strategies, in order to store in memory only a subset of the data. Others are based on the partition of the whole dataset in several independent batches for separate processing and the merging of the result in a consensuated model. Some methodologies assume that the data is a continuous stream and has to be processed on-line or in successive batches. Also, these techniques are integrated in different ways depending on the model that is used for the clustering process (prototype based, density based, ...). This large variety of approaches makes necessary to define their characteristics and to organize them in a coherent way.

The outline of this paper is as follows. In section 2 some preprocessing techniques available for dimensionality reduction will be reviewed. In section 3 the different paradigms for clustering data will be presented, analyzing their capability for processing large quantities of data. Section 4 will discuss the different strategies used for scaling clustering algorithms. Section 5 describe some algorithms that use these scalability strategies. Finally, section 6 will outline some conclusions.

## 2 Preprocessing: Dimensionality reduction

Before applying the specific mining task that has to be performed on a dataset, several preprocessing steps can be done. The first goal of the preprocessing step is to assure the quality of the data by reducing the noisy and irrelevant information that it could contain. The second goal is to reduce the size of the dataset, so the computational cost of the discovery task is also reduced.

There are two dimensions that can be taken in account when reducing the size of the dataset. The first one is the number of instances. This problem can be addressed by sampling techniques when it is clear that a smaller subset of the data holds the same information that the whole dataset. Not in all application it is the case, and sometimes the specific goal of the mining process is to find specific groups of instances with low frequency, but of high value. This data could be discarded by the sampling process, making unfruitful the process. In other applications, the data is a stream, this circumstance makes more difficult the sampling process or carries the risk of losing important information from the data if its distribution changes over time.

With dimensionality reduction techniques, the number of attributes of the dataset also can be addressed. There are several areas related to the transformation of a dataset from the original representation to a representation with a reduced set of features. The goal is to obtain a new dataset that preserves, up to a level, the original structure of the data, so its analysis will result in the same or equivalent patterns present in the original data. Broadly, there are two kinds of methods for reducing the attributes in a dataset, feature selection and feature extraction.

Most of the research on feature selection is related to supervised learning [14]. More recently, methods for unsupervised learning have been appearing in the literature [4], [11], [24], [26]. These methods can be divided on *filters*, that use characteristics of the features to determine their salience so the more relevant ones can be kept, and *wrappers*, that involve the exploration of the subset of features and a clustering algorithm to evaluate the quality of the partitions generated with the subset, according to a internal or external quality criteria. The main advantage of all these methods is that they preserve the original attributes, so the resulting patterns can be interpreted more easily.

Feature extraction is an area with a large number of methods. The goal is to create a smaller new set of attributes that maintains the patterns present in the data. This reduction process is used frequently to visualize the data to help with the discovery process. These methods generate new attributes that are linear or non linear combinations of the original attributes. The most popular method that obtains a linear transformation of the data is Principal Component Analysis [10]. This transformation results in a set of orthogonal dimensions that account for the variance of the dataset. It is usual for only a small number of these dimensions to hold most of the variance of the data, so with only this subset should be enough to discover the patterns in the data. Nonlinear feature extraction methods have been becoming more popular because of the ability to uncover more complex patterns in the data. Popular examples of these methods include the kernelized version of PCA [19] and methods based on manifold learning like ISOMAP [21] or Locality Linear Embedding [18]. A mayor drawback these methods is their computational cost. Most of them include some sort of matrix factorization and scale poorly.

## 3 Clustering algorithms

The clustering task can be defined as a process that, using the intrinsic properties of a dataset $\mathcal{X}$, uncovers a set of partitions that represents its inherent structure. It is, thus, an usupervised task, that relies in the patterns that present the values of the attributes that describe the dataset. The partitions can be either nested, so a hierarchical structure is represented, or disjoint partitions with or without overlapping.

There are several approaches to obtain a partition from a dataset, depending on the characteristics of the data or the kind of the desired partition. Broadly these approaches can be divided in:

- Hierarchical algorithms, that result in a nested set of partitions, representing the hierarchical structure of the data. These methods are usually based on a matrix of distances/similarites and a recursive divisive or agglomerative strategy.
- Partitional algorithms, that result in a set of disjoint or overlapped partitions. There is a more wide variety of methods of this kind, depending on the model used to represent the partitions or the discovery strategy used. The more representative ones include algorithms based on prototypes or probabilistical models, based on the discovery of dense regions and based on the partition of the space of examples into a multidimensional grid.

In the following sections the main characteristics of these methods will be described with an outline of the main representative algorithms.

### 3.1 Hierarchical clustering

Hierarchical methods [5] use two strategies for building a tree of nested clusters that partitions a dataset, divisive and agglomerative. Divisive strategies begin with the entire dataset, and each iteration it is determined a way to divide the data into two partitions. This process is repeated recursively until individual examples are reached. Agglomerative strategies iteratively merge the most related pair of partitions according to a similarity/distance measure until there is only one partition. Usually agglomerative strategies are computationally more efficient.

These methods are based on a distance/similarity function that compares partitions and examples. The values of these measures for each pair of examples are stored in a matrix that is updated during the clustering process.

Some algorithms consider this matrix as a graph that is created by adding each iteration new edges in an ascending/descending order of length. In this case, a criteria determines when the addition of a new edge result in a new clusters. For example, the single link criteria defines a new cluster each time a new edge is added if that connects two disjoint groups, or the complete link criteria considers that a new cluster appears only when a the union of two disjoint groups form a clique in the graph.

Other algorithms reduce the distance/similarity matrix each iteration by merging two groups, deleting these groups from the matrix and then adding the new merged group. The distances of this new group to the remaining groups are recomputed as a combination of the distances to the two merged group. Popular choices for this combination are the maximum, minimum and mean of these distances. Also the size or the variance of the clusters can be used to weight the combination.

These variety of choices in the criteria for merging the partitions and updating the distances, creates a continuous of algorithms that can obtain very different partitions from the same data. But the main drawback of these algorithms is their computational cost. The distance matrix has to be stored and this scales quadratically with the number of examples. Also, the computational cost of these algorithms is cubic with the number of examples in the general case and it can be reduced in some particular cases to $O(n^2 \log(n))$ or even $O(n^2)$.

### 3.2 Prototype/model based clustering

Prototype and model based clustering assume that clusters fit to a specific shape, so the goal is to discover how different numbers of these shapes can explain the spatial distribution of the data.

The most used prototype based clustering algorithm is K-Means [5]. This algorithm assumes that clusters are defined by their center (the prototype) and have spherical shapes. The fitting of this spheres is done by minimizing the distances from the examples to these centers. Examples are only assigned to one cluster.

Different optimization criteria can be used to obtain the partition, but the most common one is the minimization of the sum of the euclidean distance of the examples assigned to a cluster and the centroid of the cluster. The problem can be formalized as:

$$\min_{\mathcal{C}} \sum_{\mathcal{C}_i} \sum_{x_j \in \mathcal{C}_i} \| x_j - \mu_i \|_2$$

This minimization problem is solved iteratively using a gradient descent algorithm.

Model based clustering assumes that the dataset can be fit to a mixture of probability distributions. The shape of the clusters depends on the specific probability distribution used. A common choice is the gaussian distribution. In this case, depending on the choice about if covariances among attributes are modelled or not, the clusters correspond to arbitrarily oriented ellipsoids or spherical shapes. The model fit to the data can be expressed in general as:

$$P(x|\theta) = \sum_{i=1}^{K} P(w_i) P(x|\theta_i, w_i)$$

Being $K$ the number of clusters, with $\sum_{i=1}^{K} P(w_i) = 1$. This model is usually fit using the Expectation-Maximization algorithm (EM), assigning for each example a probability to each cluster.

The main limitation of all these methods is to assume that the number of partition is known. Both types of algorithms need to have all the data in memory for performing the computations, so their spatial needs scale linearly with the size of the dataset. The storage of the model is just a fraction of the size of the dataset for prototype based algorithms, but for model base algorithms depends on the number of parameters needed to estimate the probability distributions, this number can grow quadratically with the number of attributes if, for example, gaussian distributions with full covariance matrices are used. The computational time cost for the prototype based algorithms depends on the number of examples ($n$), the number of attributes ($d$), the number of clusters ($k$) and the number of iterations needed for convergence ($i$), so it is proportional to $O(ndki)$. The number of iterations depends on the dataset, but it is bounded by a constant. For the model based algorithms, each iteration has to estimate all the parameters of the distribution of the model for all instances, so the computational time depends on the number of parameters, that in the case of full covariance estimation, each iteration results in a total time complexity of $O(nd^2k)$

## 3.3 Density based clustering

Density based clustering does not assumes an specific shape for the clusters or that the number of clusters is known. The goal is to uncover areas of high density in the space of examples. There are different strategies to find the dense areas of a dataset, but the usual methods are derived from the works of the algorithm DBSCAN [6].

This algorithm is based on the idea of *core points*, that constitute the examples that belong to the interior of the clusters, and the neighborhood relations of this points with the rest of the examples. The $\varepsilon$-neighborhood of an example ($N_\varepsilon(x)$) is defined as the set of instances that are at a distance less than $\varepsilon$ to a given instance. A core point is defined as the examples that have more than a certain number of elements ($MinPts$) in its $\varepsilon$-neighborhood. From this neighborhood sets, different reachability relations are defined allowing to connect density areas defined by these core points. A cluster is defined as all the core points that are connected by this reachability relations and the points that belong to their neighborhood.

The key point of this algorithm is the choice of the $\varepsilon$ and $MinPts$ parameters, the algorithm OPTICS [2] is an extension of the original DBSCAN that uses heuristics to find good values for these parameters.

The main drawback of this methods comes from the cost of finding the nearest neighbors for an example. Indexing data structures can be used to reduce the computational time, but these structures degrade with the number of dimensions to a linear search. This makes the computational time of these algorithms proportional to the square of the number of examples for datasets with a large number of dimensions.

### 3.4 Grid based clustering

Grid based clustering is another approach to finding dense areas of examples. The basic idea is to divide the space of instances in hyperrectangular cells by discretizing the attributes of the dataset. In order to avoid to generate a combinatorial number of cells, different strategies are used. It has to be noticed the fact that, the maximum number of cells that contain any example is bounded by the size of the dataset. Clusters of arbitrary shapes can be discovered using these algorithms.

The different algorithms usually rely on some hierarchical strategy to build the grid top down or bottom up. For example, the algorithm STING [23] assumes that the data has a spatial relation and, beginning with one cell, recursively partitions the current level into four cells chosen by the density of the examples. Each cell is summarized by the sufficient statistics of the examples it contains. The algorithm of CLIQUE [1] uses a more general approach. Assumes that the attributes of the dataset have been are discretized and the one dimensional dense cells for each attribute can be identified. This cells are merged attribute by attribute in a bottom up fashion, considering that a merging only can be dense if the cells of the attributes that compose the merge are dense. This antimonotonic property allows to prune the space of possible cells. Once the cells are identified, the clusters are formed by finding the connected components in the graph defined by the adjacency relations of the cells.

These methods can usually scale well, but it depends on the granularity of the discretization of the space of examples. The strategies used to prune the search space allow to largely reduce the computational cost, that scales on the number of examples and a quadratic factor in the number of attributes.

### 3.5 Other approaches

There are several other approaches that use other criteria for obtaining a set of clusters from a dataset. Two methods have gained popularity in the latest years: spectral clustering and affinity clustering.

Spectral clustering methods [15] define a Laplacian matrix from the similarity matrix of the dataset that can be used to define different clustering algorithms. Assuming that the Laplacian matrix represent the neighborhood relationships among examples, the eigenvectors of this matrix can be used as a dimensionality reduction method, transforming the data to a new space where traditional clustering algorithms can be applied. This matrix can also be used to solve a min-cut problem for the defined graph. Several objective functions have been defined for this purpose. The computational complexity of this family of methods is usually high because the computation of the eigenvectors of the Laplacian matrix is needed. This cost can be reduced by approximating the first $k$ eigenvalues of the matrix.

Affinity clustering [7] is an algorithm based on message passing. Iterativelly, the number of clusters and their representatives are determined by refining a pair of measures, *responsibility*, that accounts for the suitability of an exemplar for being a representative of a cluster and *availability*, that accounts for the evidence that certain point is the representative of other example. This two measures are linked by a set of equations and are initialized using the similarity among the examples. The algorithm recomputes this measures each iteration until a stable set of clusters appear. The computational complexity of this method is quadratic on the number of examples

## 4 Scalability strategies

The strategies used to scale clustering algorithms range from general strategies that can be adapted to any algorithm to specific strategies that exploit the characteristics of the algorithm in order to reduce its computational cost.

Some of the strategies are also dependent on the type of data that is used. For instance, only clustering algorithms that incrementally build the partition can be used for data streams. For this kind of datasets it means that the scaling strategy has to assume that the data will be processed continuously and only one pass through the data will be allowed. For applications where the whole dataset can be stored in secondary memory, other possibilities are also available.

The different strategies applied for scalability are not disjoint and several strategies can be used in combination. These strategies can be classified in:

One-pass strategies: The constraint assumed is that the data only can be processed once and in a sequential fashion. A new example is integrated in the model each iteration. Depending on the type of the algorithm a data structure can be used to efficiently determine how to perform this update. This strategy does not only apply to data streams and can be actually used for any dataset.

Summarization strategies: It is assumed that all the examples in the dataset are not needed for obtaining the clustering, so an initial preprocess of the data can be used to reduce its size by combining examples. The preprocess results in a set of representatives of groups of examples that fits in memory. The representatives are then processed by the clustering algorithm.

Sampling/batch strategies: It is assumed that processing samples of the dataset that fit in memory allows to obtain an approximation of the partition of the whole dataset. The clustering algorithm generates different partitions that are combined iterativelly to obtain the final partition.

Approximation strategies: It is assumed that certain computations of the clustering algorithm can be approximated or reduced. These computations are mainly related with the distances among examples or among the examples and the cluster prototypes.

Divide and conquer strategies: It is assumed that the whole dataset can be partitioned in roughly independent datasets and that the combination/union of the results for each dataset approximates the true partition.

## 4.1 One-pass strategies

The idea of this strategy is to reduce the number of scans of the data to only one. This constraint may be usually forced by the circumstance that the dataset can not fit in memory and it has to be obtained from disk. Also the constraint could be imposed by a continuous process that does not allow to store all the data before processing it.

Sometimes this strategy is used to perform a preprocess of the dataset. This results in two stages algorithms, a first one that applies the one-pass strategy and a second one that process in memory a summary of the data obtained by the first stage.

The assumption of the first stage is that a simple algorithm can be used to obtain a coarse representation of the clusters in the data and that these information will be enough to partition the whole dataset.

Commonly this strategy is implemented using the leader algorithm. This algorithm does not provide very good clusters, but can be used to estimate densities or approximate prototypes, reducing the computational cost of the second stage.

## 4.2 Summarization Strategies

The purpose of this strategy is to obtain a coarse approximation of the data without losing the information that represent the different densities of examples. This summarization strategy assumes that there is a set of sufficient quantities that can be computed from the data, capable of representing their characteristics. For instance, by using sufficient statistics like mean and variance.

The summarization can be performed single level, as a preprocess that is feed to a cluster algorithm able to process summaries instead of raw examples, or also can be performed in a hierarchical fashion.

This hierarchical scheme can reduce the computational complexity by using a multi level clustering algorithm or can be used as an element of a fast indexing structure that reduces the cost of obtaining the first level summarization.

### 4.3 Sampling/batch strategies

The purpose of sampling and batch strategies is to allow to perform the processing in main memory for a part of the dataset.

Sampling assumes that only a random subset or subsets of the data are necessary to obtain the model for the data and that the complete dataset is available from the beginning. The random subsets can be or not disjoint. If more than one sample of the data is processed, the successive samples are integrated with the current model. This is usually done using an algorithm able to process raw data and cluster summaries. The algorithms that use this strategy do not process all the data, so they scale on the size of the sampling and not on the size of the whole dataset.

The use of batches assume that the data can be processed sequentially and that after applying a clustering algorithm to a batch, the result can be merged with the results from previous batches. This processing assumes that data is available sequentially as in a data stream and that the batch is complete after observing an amount of data that fits in memory.

### 4.4 Approximation strategies

These strategies assume that some computations can be saved or approximated with reduced or null impact on the final result. The actual approximation strategy is algorithm dependent, but usually the most costly part of clustering algorithms corresponds to distance computation among instances or among instances and prototypes. This circumstance focus these strategies particularly on hierarchical, prototype based and some density based algorithms, because they use distances to decide how to assign examples to partitions.

For example, some of these algorithms are iterative and the decision about what partition is assigned to an example does not change after a few iterations. If this can be determined at an early stage, all these distance computations can be avoided in successive iterations.

This strategy is usually combined with a summarization strategy where groups of examples are reduced to a point that is used to decide if the decision can be performed using only that point or the distances to all the examples have to be computed.

### 4.5 Divide and conquer strategies

This is a general strategy applied in multiple domains. The principle is that data can be divided in multiple independent datasets and that the clustering results can be then merged on a final model. This strategy rely sometimes on a hierarchical scheme to reduce the computational cost of merging all the independent models. Some strategies assume that each independent clustering represent a view of the model, being the merge a consensus of partitions. The approach can also result on almost independent models that have to be joined, in this case the problem to solve is how to merge the parts of the models that represent the same clusters.

## 5 Algorithms

All these scalability strategies have been implemented in several algorithms that represent the full range of different approaches to clustering. Usually more than one strategy is combined in an algorithm to take advantage of the cost reduction and scalability properties. In this section, a review of a representative set of algorithms and the use of these strategies is presented.

## 5.1 Scalable hierarchical clustering

The main drawback of hierarchical clustering is its high computational cost (time $O(n^2)$, space $O(n^2)$ ) that makes it impractical for large datasets. The proposal in [17] divides the clustering process in two steps. First a one pass clustering algorithm is applied to the dataset, resulting in a set of cluster summaries that reduce the size of the dataset. This new dataset fits in memory and can be processed using a single link hierarchical clustering algorithm.

For the one-pass clustering step, the leader algorithm is used. This algorithm has as parameter ($d$), the maximum distance between example and cluster prototype. The processing of each example follows the rule, if the nearest existing prototype is closer than $d$, it is included in that cluster and its prototype recomputed, otherwise, a new cluster with the example is created. The value of the parameter is assumed to be known or can be estimated from a sample of the dataset. The time complexity of this algorithm is $O(nk)$ being $k$ the number of clusters obtained using the parameter $d$.

The first phase of the proposed methodology applies the leader algorithm to the dataset using as a parameter half the estimated distance between clusters ($h$). For the second stage, the centers of the obtained clusters are merged using the single-link algorithm until the distance among clusters is larger than $h$.

The clustering obtained this way is not identical to the resulting from the application of the single-link algorithm to the entire dataset. To obtain the same partition, an additional process is performed. During the merging process, the clusters that have pairs of examples at a distance less than $h$ are also merged. For doing this, only the examples of the clusters that are at a distance less than $2h$ have to be examined. The overall complexity of all three phases is $O(nk)$, that corresponds to the complexity of the first step. The single-link is applied only to the cluster obtained by the first phase, reducing its time complexity to $O(k^2)$, being thus dominated by the time of the leader algorithm.

## 5.2 Rough-DBSCAN

In [22] a two steps algorithm is presented. The first step applies a one pass strategy using the leader algorithm, just like the algorithm in the previous section. The application of this algorithm results in an approximation of the different densities of the dataset. This densities are used in the second step, that consists in a variation of the density based algorithm DBSCAN.

This method uses a theoretical result that bounds the maximum number of leaders obtained by the leader algorithm. Given a radius $\tau$ and a closed and bounded region of space determined by the values of the features of the dataset, the maximum number of leaders $k$ is bounded by:

$$k \leq \frac{V_S}{V_{\tau/2}}$$

being $V_S$ the volume of the region $S$ and $V_{\tau/2}$ the volume of a sphere of radius $\tau/2$. This number is independent of the number of examples in the dataset and the data distribution.

For the first step, given a radius $\tau$, the result of the leader algorithm is a list of leaders ($\mathcal{L}$), their followers and the count of their followers. The second step applies the DBSCAN algorithm to the set of leaders given an $\epsilon$ and a $MinPts$ parameters.

The count of followers is used to estimate the count of examples around a leader. Different estimations can be derived from this count. First it is defined $\mathcal{L}_l$ as the set of leaders at a distance less or equal than $\epsilon$ to the leader $l$:

$$\mathcal{L}_l = \{l_j \in \mathcal{L} \mid \|l_j - l\| \leq \epsilon\}$$

The measure $roughcard(N_\epsilon(l, \mathcal{D}))$ is defined as:

$$roughcard(N_\epsilon(l, \mathcal{D})) = \sum_{l_i \in \mathcal{L}_l} count(l_i)$$

approximating the number of examples less than a distance $\epsilon$ to a leader. Alternate counts can be derived as upper and lower bounds of this count using $\epsilon + \tau$ (upper) or $\epsilon - \tau$ (lower) as distance.

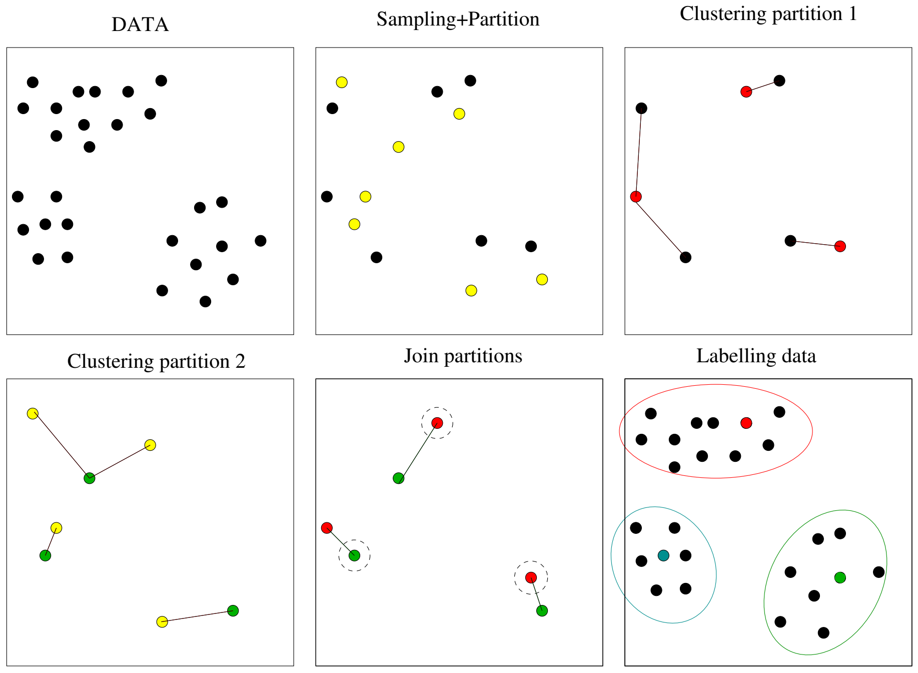

Fig. 1: CURE

From this counts it can be determined if a leader is dense or not. Dense leaders are substituted by their followers, non dense leaders are discarded as outliers. The final result of the algorithm is the partition of the dataset according to the partition of the leaders.

The computational complexity of this algorithm is for the first step $O(nk)$, being $k$ the number of leaders, that does not depend on the number of examples $n$, but on the radius $\tau$ and the volume of the region that contains the examples. For the second step, the complexity of the DBSCAN algorithm is $O(k^2)$, given that the number of leaders will be small for large datasets, the cost is dominated by the cost of the first step.

## 5.3 CURE

CURE [9] is a hierarchical agglomerative clustering algorithm. The main difference with the classical hierarchical algorithms is that it uses a set of examples to represent the clusters, allowing for non spherical clusters to be represented. It also uses a parameter that shrinks the representatives towards the mean of the cluster, reducing the effect of outliers and smoothing the shape of the clusters. Its computational cost is $O(n^2\log(n))$

The strategy used by this algorithm to attain scalability combines a divide an conquer and a sampling strategy. The dataset is first reduced by using only a sample of the data. Chernoff bounds are used to compute the minimum size of the sample so it represents all clusters and approximates adequately their shapes.

In the case that the minimum size of the sample does not fit in memory a divide and conquer strategy is used. The sample is divided in a set of disjoint batches of the same size and clustered until a certain number of clusters is achieved or the distance among clusters is less than an specified parameter. This step has the effect of a pre-clustering of the data. The clusters representatives from each batch are merged and the same algorithm is applied until the desired number of clusters is achieved. A representation of this strategy appears in figure 1. Once the clusters are obtained all the dataset is labeled according to the nearest cluster. The complexity of the algorithm is $O(\frac{n^2}{p}\log(\frac{n}{p}))$, being $n$ the size of the sample and $p$ the number of batches used.

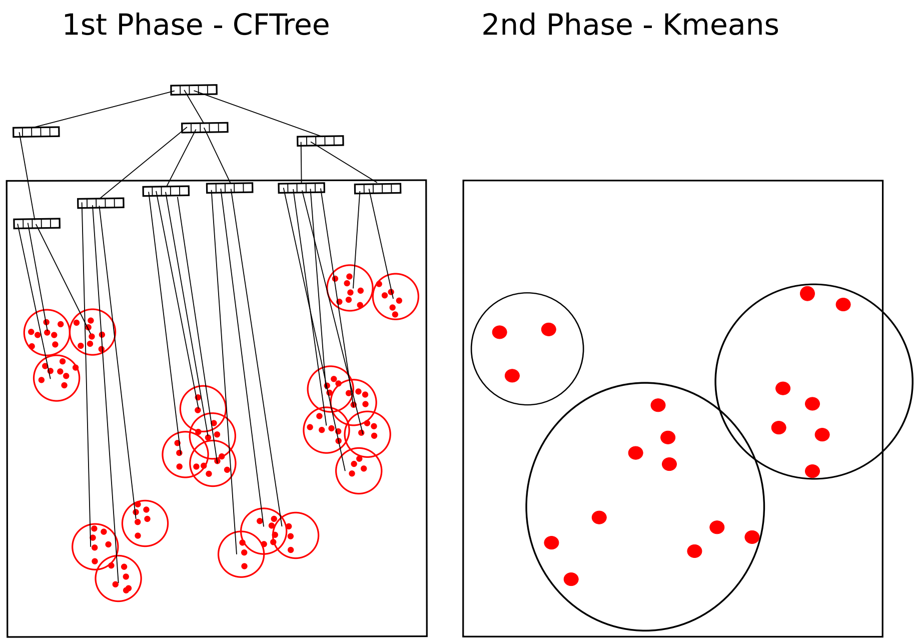

Fig. 2: BIRCH algorithm

### 5.4 BIRCH

BIRCH [27] is a multi stage clustering algorithm that bases its scalability in a first stage that incrementally builds a pre-clustering of the dataset. The first stage combines a one pass strategy and a summarization strategy that reduces the actual size of the dataset to a size that fits in memory.

The scalability strategy relies on a data structure named Clustering Feature tree (CF-tree) that stores information that summarizes the characteristics of a cluster. Specifically, the information in a node is the number of examples, the sum of the examples values and the sum of their square values. From these values other quantities about the individual clusters can be computed, for instance, the centroid, the radius of the cluster, its diameter and quantities relative to pairs of clusters, as the inter-cluster distance or the variance increase.

A CF-tree (figure 2) is a balanced n-ary tree that contains information that represents probabilistic prototypes. Leaves of the tree can contain as much as $l$ prototypes and their radius can not be more than $t$. Each non terminal node has a fixed branching factor ($b$), each element is a prototype that summarizes its subtree. The choice of these parameters is crucial, because it determines the actual available space for the first phase. In the case of selecting wrong parameters, the CF-tree can be dynamically compressed by changing the parameters values (basically $t$). In fact, $t$ determines the granularity of the final groups.

The first phase of BIRCH inserts sequentially the examples in the CF-tree to obtain a set of clusters that summarizes the data. For each instance, the tree is traversed following the branch of the nearest prototype of each level, until a leave is reached. Once there, the nearest prototype from the leave to the example is determined. The example could be introduced in this prototype or a new prototype could be created, depending whether the distance is greater or not than the value of the parameter $t$. If the current leave has not space for the new prototype (already contains $l$ prototypes), the algorithm proceeds to create a new terminal node and to distribute the prototypes among the current node and the new leaf. The distribution is performed choosing the two most different prototypes and dividing the rest using their proximity to these two prototypes. This division will create a new node in the ascendant node. If the new node exceeds the capacity of the father, it will be split and the process will continue upwards until the root of the tree is reached if necessary. Additional merge operations after completing this process could be performed to compact the tree.

For the next phase, the resulting prototypes from the leaves of the CF tree represent a coarse vision of the dataset. These prototypes are used as the input of a clustering algorithm. In the original

**Algorithm 1** OptiGrid algorithm

```
Given: number of projections k, number of cutting planes q, min cutting quality min_c_q,
       data set X
Compute a set of projections $P = \{P_1, ..., P_k\}$
Project the dataset X wrt the projections $\{P_1(X), ..., P_k(X)\}$
BestCuts ← ∅, Cut ← ∅
for $i \in 1..k$ do
    Cut ← ComputeCuts($P_i(X)$)
    for c in Cut do
        if CutScore(c)> min_c_q then BestCuts.append(c)
    end
end
if BestCuts.isEmpty() then return X as a cluster
BestCuts ← KeepQBestCuts(BestCuts,q)
Build the grid for the q cutting planes
Assign the examples in X to the cells of the grid
Determine the dense cells of the grid and add them to the set of clusters $C$
foreach cluster $cl \in C$ do
    apply OptiGrid to $cl$
end
```

algorithm, single link hierarchical clustering is applied, but also K-means clustering could be used. The last phase involves labeling the whole dataset using the centroids obtained by this clustering algorithm. Additional scans of the data can be performed to refine the clusters and detect outliers.

The actual computational cost of the first phase of the algorithm depends on the chosen parameters. Chosen a threshold $t$, considering that $s$ is the maximum number of leaves that the CF-tree can contain, also that the height of the tree is $\log_b(s)$ and that at each level $b$ nodes have to be considered, the temporal cost is $O(nb\log_b(s))$. The temporal cost of clustering the leaves of the tree depends on the algorithm used, for hierarchical clustering it is $O(s^2)$. Labeling the dataset has a cost $O(nk)$, being $k$ the number of clusters.

## 5.5 OptiGrid

OptiGrid [12] presents an algorithm that divides the space of examples in an adaptive multidimensional grid that determines dense regions. The scalability strategy is based on recursive divide and conquer. The computation of one level of the grid determines how to divide the space on independent datasets. These partitions can be divided further until no more partitions are possible.

The main element of the algorithm is the computation of a set of low dimensional projections of the data that are used to determine the dense areas of examples. These projections can be computed using PCA or other dimensionality reduction algorithms and can be fixed for all the iterations. For a projection, a fixed number of orthogonal cutting planes are determined from the maxima and minima of the density function computed using kernel density estimation or other density estimation method. These cutting planes are used to compute a grid. The dense cells of the grid are considered clusters at the current level and are recursively partitioned until no new cutting planes can be determined given a quality threshold. A detailed implementation is presented in algorithm 1

For the computational complexity of this method. If the projections are fixed for all the computations, the first step can be obtained separately of the algorithm and is added to the total cost. The actual cost of computing the projections depends on the method used. Assuming axis parallel projections the cost for obtaining $k$ projections for $N$ examples is $O(Nk)$, $O(Ndk)$ otherwise, being $d$ the number of dimensions. Computing the cutting planes for $k$ projections can be obtained also in $O(Nk)$. Assigning the examples to the grid depends on the size of the grid and the insertion time for

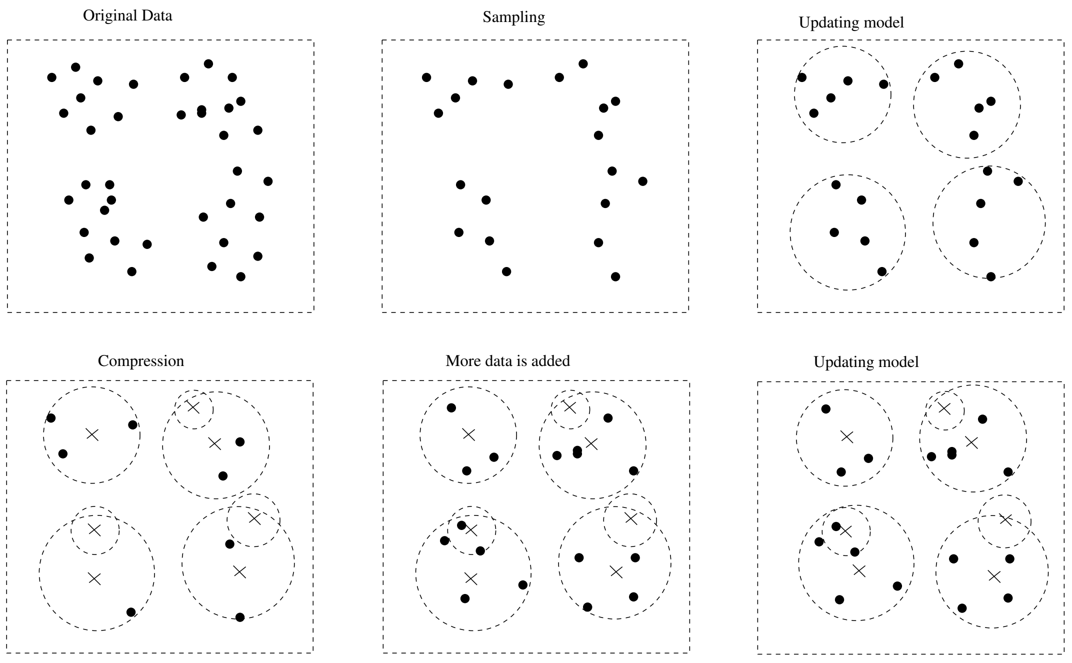

Fig. 3: Scalable K-means

the data structure used to store the grid. For $q$ cutting planes and assuming a logarithmic insertion time structure, the cost of assigning the examples has a cost of $O(Nq\min(q, \log(N)))$ considering axis parallel projections and $O(Nqd\min(q, \log(N)))$ otherwise. The number of recursions of the algorithm is bound by the number of clusters in the dataset that is a constant. Considering that $q$ is also a constant, this gives a total complexity that is bounded by $O(Nd\log(N))$

## 5.6 Scalable K-means

This early algorithm for clustering scalability presented in [3] combines a sampling strategy and a summarization strategy. The main purpose of this algorithm is to provide an on-line and anytime version of the K-means algorithm that works with a pre-specified amount of memory.

The algorithm repeats the following cycle until convergence:

1. Obtain a sample of the dataset that fits in the available memory
2. Update the current model using K-means
3. Classify the examples as:
   (a) Examples needed to compute the model
   (b) Examples that can be discarded
   (c) Examples that can be compressed using a set of sufficient statistics as fine grain prototypes

The discarding and compressing of part of the new examples allows to reduce the amount of data needed to maintain the model each iteration.

The algorithm divides the compression of data in two differentiated strategies. The first one is called primary compression, that aims to detect those examples that can be discarded. Two criteria are used for this compression, the first one determines those examples that are closer to the cluster centroid than a threshold. These examples are not probably going to change their assignment in the future. The second one consist in perturbing the centroid around a confidence interval of its values. If an example does not change its current cluster assignment, it is considered that future modifications of the centroid will still include the example.

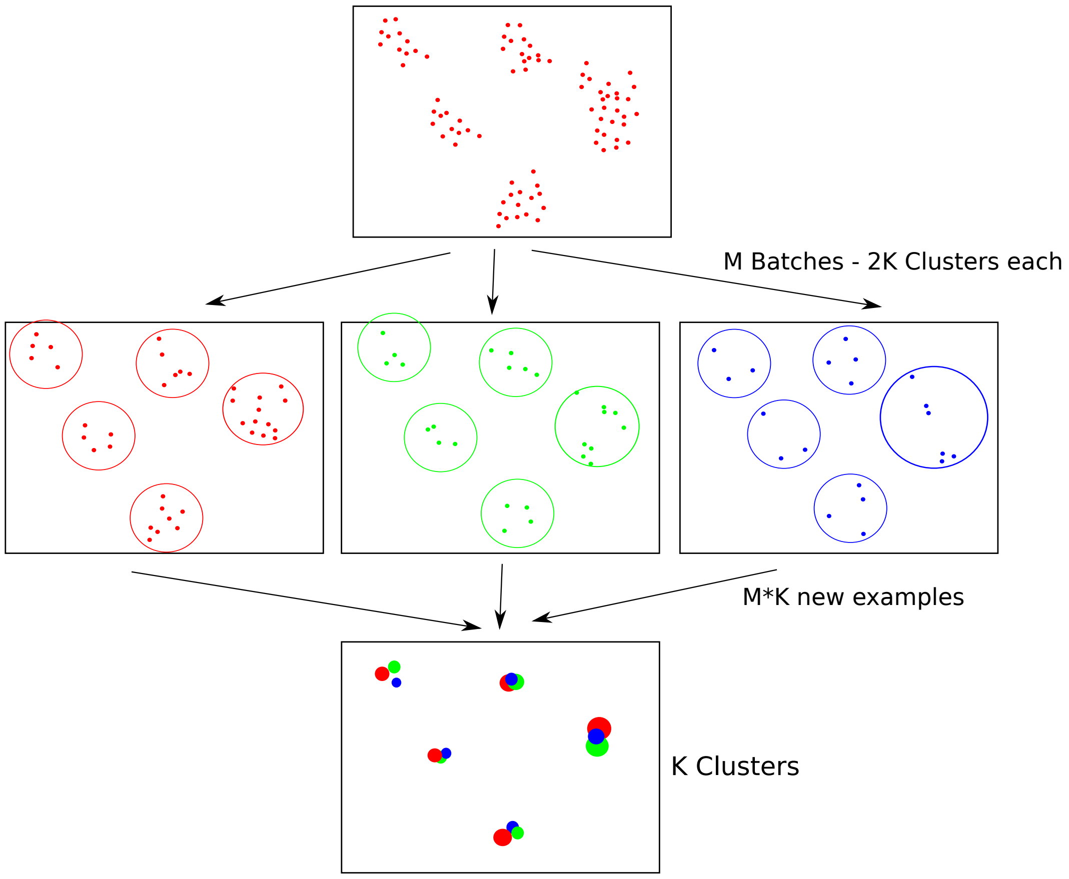

Fig. 4: STREAM LSEARCH

The second strategy is called secondary compression, that aims to detect those examples that can not be discarded but form a compact subcluster. In this case, all the examples that form these compact subclusters are summarized using a set of sufficient statistics. The values used to compute that sufficient statistics are the same used by BIRCH to summarize the dataset.

The algorithm used for updating the model is a variation of the K-means algorithm that is able to treat single instances and also summaries. The temporal cost of the algorithm depends on the number of iterations needed until convergence as in the original K-means, so the computational complexity is $O(kni)$, being $k$ the number of clusters, $n$ the size of the sample in memory, and $i$ the total number of iterations performed by all the updates.

## 5.7 STREAM LSEARCH

The STREAM LSEARCH algorithm [8] assumes that data arrives as a stream, so holds the property of only examining the data once. The algorithm process the data in batches obtaining a clustering for each batch and merging the clusters when there is not space to store them. This merging is performed in a hierarchical fashion. The strategy of the algorithm is then a combination of one-pass strategy plus batch and summarization strategies.

The basis of the whole clustering scheme is a clustering algorithm that solves the facility location (FL) problem. This algorithm reduces a sequential batch of the data to at most $2k$ clusters, that summarize the data. These clusters are used as the input for the hierarchical merging process. The computational cost of the whole algorithm relies on the cost of this clustering algorithm. This algorithm finds a set of between $k$ and $2k$ clusters that optimizes the FL problem using a binary search strategy. An initial randomized procedure computes the clusters used as initial solution. The cost of this algorithm is $O(nm + nk\log(k))$ being $m$ the number of clusters of the initial solution, $n$ the number of examples and $k$ the number of clusters.

The full algorithm can be outlined as:

**Algorithm 2** Mini Batch K-Means algorithm

```
Given: k, mini-batch size b, iterations t, data set X
Initialize each c ∈ C with an x picked randomly from X
v ← 0
for $i$ ← $1$ to $t$ do
    M ← b examples picked randomly from X
    for $x \in M$ do
        d[x] ← f(C,x)
    end
    for $x \in M$ do
        c ← d[x]
        v[c] ← v[c] + 1
        $\eta \leftarrow \frac{1}{v[c]}$
        c ← (1-$\eta$)c+$\eta$x
    end
end
```

1. Input the first $m$ points; use the base clustering algorithm to reduce these to at most $2k$ cluster centroids. The number of examples at each cluster will act as the weight of the cluster.
2. Repeat the above procedure until $\frac{m^2}{2k}$ examples have been processed so we have $m$ centroids
3. Reduce them to $2k$ second level centroids
4. Apply the same criteria for each existing level so after having $m$ centroids at level $i$ then $2k$ centroid at level $i + 1$ are computed
5. After seen all the sequence (or at any time) reduce the $2k$ centroids at top level to $k$ centroids

The number of centroids to cluster is reduced geometrically with the number of levels, so the main cost of the algorithm relies on the first level. This makes the time complexity of the algorithm $O(nk \log(nk))$, while needing only $O(m)$ space.

## 5.8 Mini batch K-means

Mini Batch K-means [20] uses a sampling strategy to reduce the space and time that K-means algorithm needs. The idea is to use small bootstrapped samples of the dataset of a fixed size that can be fit in memory. Each iteration, the sample is used to update the clusters. This procedure is repeated until the convergence of the clusters is detected or a specific number of iterations is reached.

Each mini batch of data updates the cluster prototypes using a convex combination of the attribute values of the prototypes and the examples. A learning rate that decreases each iteration is applied for the combination. This learning rate is the inverse of number of examples that have been assigned to a cluster during the process. The effect of new examples is reduced each iteration, so convergence can be detected when no changes in the clusters occur during several consecutive iterations. A detailed implementation is presented in algorithm 2.

The mayor drawback of this algorithm is that the quality of the clustering depends on the size of the batches. For very large datasets, the actual size for a batch that can be fit in memory can be very small compared with the total size of the dataset. The mayor advantage is the simplicity of the approach. This same strategy is also used for scaling up other algorithms as for example backpropagation in artificial neural networks.

The complexity of the algorithm depends on the number of iterations needed for convergence ($i$), the size of the samples ($n$), and the number of clusters ($k$) so it is bound by $O(kni)$.

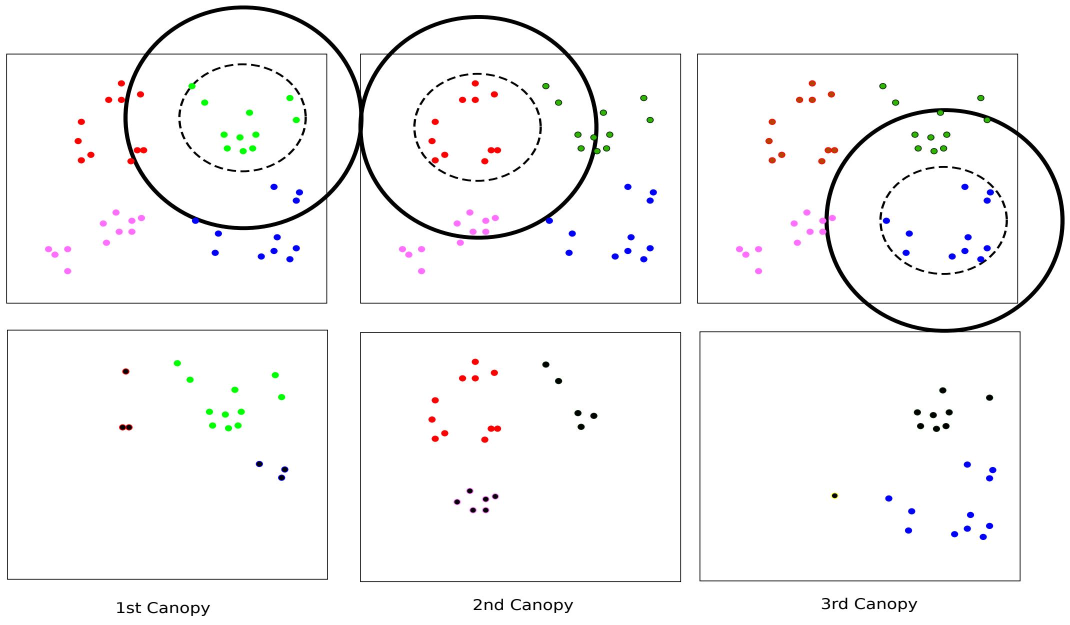

Fig. 5: Canopy clustering

## 5.9 Canopy clustering

Canopy clustering [16] uses a divide and conquer and an approximation strategies to reduce the computational cost. It also uses a two phases clustering algorithm, that is implemented using the well known mapreduce paradigm for concurrent programming.

The first stage divides the whole dataset in a set of overlapping batches called *canopies*. The computation of these batches depends on a cheap approximate distance that determines the neighborhood of a central point given two distance thresholds. The smaller distance ($T_2$) determines the examples that will belong exclusively to a canopy. The larger distance ($T_1$) determines the examples that can be shared with other canopies. The values of these two distance thresholds can be manually determined or computed using crossvalidation.

To actually reduce the computational cost of distance computation the distance function used in this first phase should be cheap to compute. The idea is to obtain an approximation of the densities in dataset. The specific distance depends on the characteristics of the attributes in the dataset, but it is usually simple to obtain such a function by value discretization or using locality sensitive hashing.

The computation of the canopies proceeds as follows: One example is randomly picked as the center of a canopy from the dataset, all the examples that are at a distance less than $T_2$ are assigned to this canopy and can not be used as centers in the future iterations. All the examples that are at a distance less than $T_1$ are included in the canopy but can be used as centers in the future. The process is repeated until all the examples have been assigned to a canopy. In figure 5 can be seen a representation of this process.

The second stage of the algorithm consist in clustering all the canopies separately. For this process, different algorithms can be used, for example agglomerative clustering, expectation maximization (EM) for gaussian mixtures or K-means. Also different strategies can be used for applying these algorithms. For example, for K-means or EM the number of prototypes for a canopy can be fixed at the beginning, using only the examples inside the canopy to compute them, saving this way many distance computations. Other alternative is to decide the number of prototypes globally, so they can move among canopies and be computed not only using the examples inside a canopy, but also using the means of the nearest canopies.

These different alternatives make difficult to give a unique computational complexity for all the process. For the first stage, the data has to be divided in canopies, this computational cost depends on the parameters used. The method used for obtaining the canopies is similar to the one used by the leader algorithm, this means that equivalently as was shown in 5.2, the number of partitions obtained

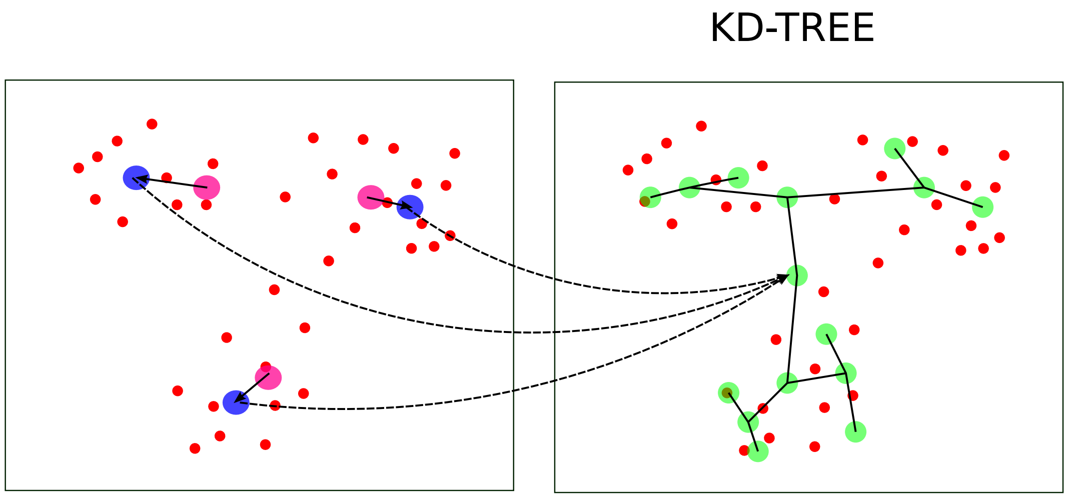

Fig. 6: Indexed K-means

does not depend on the total number of examples ($n$), but on the volume defined by the attributes and the value of parameter $T_2$. Being $k$ this number of canopies, the computational cost is bounded by $O(nk)$. The cost of the second stage depends on the specific algorithm used, but the number of distance computations needed for a canopy will be reduced in a factor $\frac{n}{k}$, so for example if single link hierarchical clustering is applied the total computational cost of applying this algorithm to $k$ canopies will be $O(\frac{n}{k}^2)$.

## 5.10 Indexed K-means

The proposal in [13] relies in an approximation strategy. This strategy is applied to the K-means algorithm. One of the computations that have most impact in the cost of this algorithm is that, each iteration all the distances from the examples to the prototypes have to be computed. One observation about the usual behavior of the algorithms is that, after some iterations, most of the examples are not going to change their cluster assignment for the remaining iterations, so computing their distances increases the cost, without having an impact in the decisions of the algorithm.

The idea is to reduce the number of distance computations by storing the dataset in an intelligent data structure that allows to determine how to assign them to the cluster prototypes. This data structure is a kd-tree, a binary search tree that splits the data along axis parallel cuts. Each level can be represented by the centroid of all the examples assigned to each one of the two partitions.

In this proposal, the K-means algorithm is modified to work with this structure. First, a kd-tree is built using all the examples. Then, instead of computing the distance from each example to the prototypes and assigning them to the closest one, the prototypes are inserted in this kd-tree. At each level, the prototypes are assigned to the branch that has the closest centroid. When a branch of the tree has only one prototype assigned, all the examples in that branch can be assigned directly to that prototype, avoiding further distance computations. When a leave of the kd-tree is reached and still there is more that one prototype, the distances among the examples and the prototypes are computed and the assignments are decided by the closest prototype as in the standard K-means algorithm. A representation of this algorithm can be seen in figure 6.

The actual performance depends on how separated are the clusters in the data and the granularity of the kd-tree. The more separated the clusters are, the less distance computations have to be performed, as the prototypes will be assigned quickly to only one branch near to the root of the kd-tree.

The time computational cost in the worst case scenario is the same as K-means, as in this case all the prototypes will be assigned to all branches, so all distance computations will be performed. The more favorable case will be when the clusters are well separated and the number of levels in the kd-tree is logarithmic respect to the dataset size ($n$), this cost will depend also on the volume enclosed in the

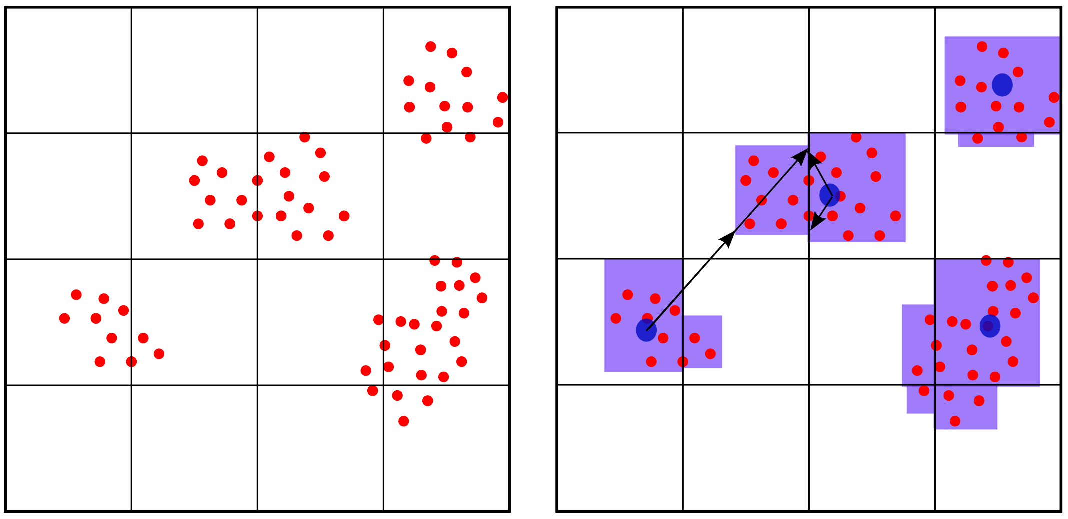

Fig. 7: Quantized K-means

leaves of the kd-tree and the number of dimensions ($d$). The computational cost for each iteration is bound by $\log(2^d k \log(n))$.

The major problem of this algorithm is that as the dimensionality increases, the benefit of the kd-tree structure degrades to a lineal search. This is a direct effect of the curse of the dimensionality and the experiments show that for a number of dimensions larger than 20 there are no time savings.

## 5.11 Quantized K-means

The proposal in [25] relies on an approximation strategy combined with a summarization strategy. The idea is to approximate the space of the examples by assigning the data to a multidimensional histogram. The bins of the histograms can be seen as summaries. This reduces the distance computations by considering all the examples inside a bin of the histogram as a unique point.

The quantization of the space of attributes is obtained by fixing the number of bins for each dimension to $\rho = \lfloor \log_m(n) \rfloor$, being $m$ the number of dimensions and $n$ the number of examples. The size of a bin is $\lambda_l = \frac{\overline{p_l} - \underline{p_l}}{\rho}$, being $\overline{p_l}$ and $\underline{p_l}$ the maximum and minimum value of the dimension $l$. All examples are assigned to a unique bin depending on the values of their attributes.

From this bins, a set of initial prototypes are computed for initializing a variation of the K-means algorithm. The computation of the initial prototypes uses the assumption that the bins with a higher count of examples are probably in the areas of more density of the space of examples. A max-heap is used to obtain these highly dense bins. Iteratively, the bin with the larger count is extracted from the max-heap and all the bins that are neighbors of this bin are considered. If the count of the bin is larger than its neighbors, it is included in the list of prototypes. All neighbor cells are marked, so they are not used as prototypes. This procedure is repeated until $k$ initial bins are selected. The centroids of these bins are used as initial prototypes.

For the cluster assignment procedure two distance functions are considered involving the distance from a prototype to a bin. The minimum distance from a prototype to a bin is computed as the distance to the nearest corner of the bin. The maximum distance from a prototype to a bin is computed as the distance to the farthest corner of the bin. In figure 7, the quantization of the dataset and these distances are represented.

Each iteration of the algorithm first computes the maximum distance from each bin to the prototypes and then it keeps the minimum of these distances as $\overline{d}(b_i, s_*)$. Then, for each prototype the minimum distance to all the bins is computed and the prototypes that are at a distance less than $\overline{d}(b_i, s_*)$ are assigned to the bins.

If only one prototype is assigned to a bin, then all its examples are assigned to the prototype without more distance computations. If there is more than one prototype assigned, the distance among the examples and the prototypes are computed and the examples are assigned to the nearest one. After the assignment of the examples to prototypes, the prototypes are recomputed as the centroid of all the examples.

Further computational improvement can be obtained by calculating the actual bounds of the bins, using the maximum and minimum values of the attributes of the examples inside a bin. This allows to obtain a more precise maximum and minimum distances from prototypes to bins, reducing the number of prototypes that are assigned to a bin.

It is difficult to calculate the actual complexity of the algorithm because it depends on the quantization of the dataset and how separated the clusters are. The initialization step that assigns examples to bins is $O(n)$. The maximum number of bins is bounded by the number of examples $n$, so at each iteration in the worst case scenario $O(kn)$ computations have to be performed. In the case that the data presents well separated clusters, a large number of bins will be empty, reducing the actual number of computations.

## 6 Conclusion

The scalability of clustering algorithms is a recent issue arisen by the need to solve unsupervised learning tasks in data mining applications. The commonly used clustering algorithms can not scale to the increased size of the datasets due to their time or space complexity. This problem opens the field for different strategies to adapt the commonly used clustering algorithms to the current needs.

This paper presents a perspective on different strategies used to scale clustering algorithm. The approaches range from the general divide and conquer scheme to more algorithm specific strategies. These strategies are used frequently in combination to obtain the different advantages that they provide. For instance, two stage clustering algorithms that apply a summarization strategy as a first stage combined with a one pass strategy.

Some algorithms implementing successfully different combinations of the presented strategies have been described in some detail, including their computational time complexity. The algorithms cover all the range of clustering paradigms including hierarchical, model based, density based and grid based algorithms.

All the discussed solutions show an evident improvement for clustering scalability. But little has been discussed about how to adjust the different parameters of these algorithms. In an scenario of very large datasets this is a challenge, and the usual trial and error does not seem an efficient approach. Further research into these methods should address this problem.

## References

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-98)*, volume 27,2 of *ACM SIGMOD Record*, pages 94–105, New York, June 1–4 1998. ACM Press.

[2] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMod-99)*, volume 28,2 of *SIGMOD Record*, pages 49–60, New York, June 1–3 1999. ACM Press.

[3] Paul S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In Rakesh Agrawal, Paul E. Stolorz, and Gregory Piatetsky-Shapiro, editors, *KDD*, pages 9–15. AAAI Press, 1998.

[4] M. Dash, K. Choi, P. Scheuermann, and H. Liu. Feature Selection for Clustering - A Filter Solution. In *ICDM*, pages 115–122, 2002.

[5] R. Dubes and A Jain. *Algorithms for Clustering Data.* PHI Series in Computer Science. Prentice Hall, 1988.

[6] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jia Wei Han, and Usama Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, page 226. AAAI Press, 1996.

[7] Frey and Dueck. Clustering by passing messages between data points. *SCIENCE: Science*, 315, 2007.

[8] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *Knowledge and Data Engineering, IEEE Transactions on*, 15(3):515 – 528, may-june 2003.

[9] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. *Inf. Syst*, 26(1):35–58, 2001.

[10] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning.* Springer, July 2001.

[11] X. He, D. Cai, and P. Niyogi. Laplacian Score for Feature Selection. In *NIPS*, 2005.

[12] Alexander Hinneburg, Daniel A Keim, et al. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *VLDB*, volume 99, pages 506–517. Citeseer, 1999.

[13] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):881 –892, jul 2002.

[14] R. Kohavi. Wrappers for Feature Subset Selection. *Art. Intel.*, 97:273–324, 1997.

[15] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[16] A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. *KDD*, pages ?–?, 2000.

[17] Bidyut Kr. Patra, Sukumar Nandi, and P. Viswanath. A distance based clustering method for arbitrary shaped clusters in large datasets. *Pattern Recognition*, 44(12):2862–2870, 2011.

[18] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2000.

[19] B. Scholkopf, A. Smola, and K. R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[20] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178. ACM, 2010.

[21] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[22] P. Viswanath and V. Suresh Babu. Rough-dbscan: A fast hybrid density based clustering method for large data sets. *Pattern Recognition Letters*, 30(16):1477 – 1488, 2009.

[23] Wei Wang, Jiong Yang, and Richard R. Muntz. STING: A statistical information grid approach to spatial data mining. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *Twenty-Third International Conference on Very Large Data Bases*, pages 186–195, Athens, Greece, 1997. Morgan Kaufmann.

[24] L. Wolf and A. Shashua. Feature Selection for Unsupervised and Supervised Inference. *Journal of Machine Learning Research*, 6:1855–1887, 2005.

[25] Zhiwen Yu and Hau-San Wong. Quantization-based clustering algorithm. *Pattern Recognition*, 43(8):2698 – 2711, 2010.

[26] H. Zeng and Yiu ming Cheung. A new feature selection method for Gaussian mixture clustering. *Pattern Recognition*, 42(2):243–250, February 2009.

[27] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: A new data clustering algorithm and its applications. *Data Min. Knowl. Discov*, 1(2):141–182, 1997.