# Clustering Algorithms for Data Mining: An Overview

Dan Ports and Sarah Lieberman
{drkp,sarahl}@mit.edu

December 15, 2004

**Abstract**

We provide a survey of current clustering algorithms as they might be applied toward data mining. We begin by presenting a list of criteria for evaluating clustering algorithms, and then consider each of the major classes of clustering algorithms and a few representative algorithms from each. In particular, hierarchical, partitioning, density-based, and grid-based algorithms are discussed. The strengths and weaknesses of each type of algorithm are evaluated, and we provide some indications of which types of data sets each algorithm is best suited for.

## Contents

# 1 Overview

As large databases of information become increasingly common, it is frequently useful to be able to identify patterns in the data. This is known as *data mining*, and is a topic of considerable interest for many companies. For example, a supermarket may wish to examine its database of sales transactions in order to identify products that are often purchased together or by the same customers, or to discover relationships between consumer demographics and their sales patterns. Network administrators may wish to monitor network traffic and discover patterns in order to create a model of normal behavior. When events occur that do not correspond to these normal behavior patterns, it may signal an intrusion, and this can be easily detected. Similarly, the government may wish to monitor association between individuals; changes in communication patterns may indicate terrorist activity.

A common technique for performing data mining is *clustering*. Clustering entails identifying groups of related data points from a large data set. For clustering purposes, the data set is often treated as a set of points in a metric space, with a distance function $d(p, q)$ reflecting the dissimilarity between two points. For data consisting of numeric values, it is often reasonable to express the data set as points in a Euclidean space and use the Euclidean distance metric to measure dissimilarity, though in some cases a different distance function may be preferable.

There is no single algorithm that is best for clustering. For a given data set, it is not clear what the optimal clustering assignment should look like, let alone how to determine it efficiently. Hence, there are a plethora of different clustering algorithms that give different results and have different runtime characteristics. The choice of which algorithm to use depends heavily on the nature of the data set in question.

## 1.1 Desirable Characteristics

Before we begin to consider the various classes of clustering algorithms currently available, it is necessary to establish some of the criteria for evaluating clustering algorithms. The following are some of the characteristics that an ideal clustering algorithm would have; clustering algorithms typically trade off some of these capabilities in order to make others possible.

- **Scalability** The data sets used in clustering algorithms are generally very large, so an algorithm should deal well with them. Often this is accomplished by using the technique of *sampling*: selecting a representative (usually random) subset of the data points and applying the algorithm to this subset rather than the full set.

- **Robustness to outliers** The data sets are often noisy, containing outliers that do not fall into any clusters. A clustering should not be affected significantly by these outliers. The algorithm should also be able to handle a small number of outliers that lie between two well-defined clusters, connecting them with a "bridge" of points, without merging the two clusters.

- **Ability to discover unusually-shaped clusters** Many algorithms assume that clusters will be spherical, consisting of all points whose distance from the cluster center is less than some value. Many data sets, however, have clusters that are non-spherical: they may be elongated, asymmetrical, or of some other unusual shape. Ideally, an algorithm should be able to discover these.

- **Minimal input parameters** An algorithm should be able to function as independently as possible, requiring little or no parameters from the user. For example, many algorithms take as a parameter $k$, the number of clusters to be identified. This must be chosen with care, because identifying too few clusters will fail to capture the useful information in the data set, while identifying too many clusters will lead to overfitting. Some algorithms, particularly density-based and grid-based ones (Sections 4 and 5), require further parameters.

- **Flexibility towards non-numerical data** Some data sets have *Boolean* or *categorical* data instead of numerical data, where the values are limited to a small set of possible options (two, in the case of Boolean data). Note that the use of an appropriately-chosen distance function allows these to be treated in largely the same way as numerical data. However, as we will see in Section 2.3, categorical data sets have other characteristics that make many algorithms not explicitly designed for them perform poorly.

### 1.2 Outline

Having established the nature of the clustering problem and the types of characteristics we would like to see in an ideal algorithm, we will now proceed to take a tour of the design space for clustering algorithms. Though there are certainly far too many clustering algorithms to consider them all here, we describe four of the most important classes of clustering algorithms, and examine two or three representative algorithms for each one. In particular, we focus not only on how each algorithm works, but on the tradeoffs it makes, and which types of data sets it is well-suited for.

Section 2 describes hierarchical algorithms, which incrementally build the desired number of clusters either by combining smaller clusters or dividing larger ones. By contrast, partitioning algorithms (Section 3) examine the data space in order to identify the clusters directly. Density-based algorithms (Section 4) are a special type of partitioning algorithms that use the density of regions rather than individual data points. Finally, grid-based algorithms (Section 5) divide the input space using a grid and process grid cells rather than data points.

## 2 Hierarchical Algorithms

Hierarchical clustering is a common technique used in clustering algorithms. Hierarchical clustering algorithms, as the name implies, treat each cluster as a hierarchy of subclusters. They can be further divided into *agglomerative* hierarchical algorithms, which start with each point in a singleton cluster and repeatedly combine clusters, and *divisive* hierarchical algorithms, which begin with every point in the same cluster and repeatedly divide clusters into smaller ones. In general, these algorithms use some sort of *linkage metric* that measures the dissimilarity between two clusters, and select the clusters to combine or divide that maximize this metric.

As we shall see, these algorithms are quite flexible with regards to different measures of similarity. This makes them applicable to a number of different situations, including cases where the data points are categorial rather than numeric. Another advantage is that the construction of clusters proceeds incrementally. Thus, if the desired number of clusters $k$ is unknown, as it often is, the algorithm can use a different termination criterion to determine when to stop joining or dividing clusters.

Hierarchical algorithms are, however, limited by the fact that the technique is fundamentally greedy: if a cluster is divided into two subclusters, the algorithms will never reconsider the assignment of points into each subcluster. It might, for example, prove optimal to first split a cluster into two subclusters $A$ and $B$, but then create a third subcluster containing the points from both $A$ and $B$ that are closest to the boundary. Hierarchical algorithms are unable to recognize these types of cases.

### 2.1 Single-link Hierarchical Clustering: SLINK

The single link technique is the classic example of a hierarchical clustering algorithms. It is an agglomerative hierarchical algorithm based on the *single-link* linkage metric. That is, it defines the similarity between two clusters as the distance between the two closest points in each cluster:

$$linkage(A, B) = \min_{\substack{x \in A \\ y \in B}} d(x, y)$$

A straightforward algorithm that implements this technique is as follows [15]: we begin by placing each element in its own cluster. Throughout the algorithm, we will maintain a (symmetric) matrix $L$ where $L_{i,j} = linkage(i, j)$, and an array $N$ where $N_i$ is the cluster most closely linked to cluster $i$, i.e. the $j$ that minimizes $linkage(i, j)$. Since the set of clusters is initially the same as the set of data points, we can initialize the arrays by setting $L_{i,j} = d(i, j)$ for each $i$ and $j$ in the data set, and iterating over each row $i$ in $L$ to find the minimum element, which gives the value of $N_i$.

Then, until the termination criterion is reached, we repeat the following procedure: we find a cluster $i$ that minimizes the linkage between it and its closest neighbor (which is $L_{i,N_i}$). Let $j = N_i$. Then we merge cluster $j$ into $i$, removing cluster $j$ and assigning its elements to $i$, and update the arrays. Now for every $x$, the new values of $L_{x,i}$ and $L_{x,j}$ are $\min(L_{x,i}, L_{x,i})$, since the closest point to $x$ in the newly-merged cluster is the closest point in either of its subclusters. Any $x$ that previously had $N_x = j$ is updated to $N_x = i$.

Note that this algorithm requires $O\left(n^2\right)$ space to store the matrix. It also requires $O\left(n^2\right)$ time, since each of the $O\left(n\right)$ merge steps requires $O\left(n\right)$ time to update two rows of $L$ and the array $N$.

The optimal implementation of this technique is the SLINK algorithm, introduced by Sibson in 1973 [18]. It is a refinement of the above procedure that requires only $O\left(n\right)$ space (but still $O\left(n^2\right)$ time).

In addition to their high runtime, single-link clustering algorithms suffer from a "chaining effect": if two clusters are connected by a chain of points, they will be merged, since the linkage between two clusters is computed by their closest points. This happens even if the clusters are large and there are only a few points connecting them — since outlier points are common in large data sets, this problem can occur frequently. [10]

This problem can be minimized by defining the linkage metric as the average distance between points in the two clusters (known as *all-points* clustering). This limits the effects of outliers, but introduces a new problem: the clusters it creates are effectively spheres around their mean, which becomes problematic if the data set is not composed of spherical clusters but irregular shapes.

### 2.2 Representative Points and Sampling: CURE

The CURE algorithm [10] is another hierarchical agglomerative algorithm that addresses the two major shortcomings of single-link clustering. It increases scalability for large data sets by using random sampling, and it provides improved handling for irregularly-shaped clusters and outliers with a new linkage metric. Rather than compute the linkage between two clusters as the distance between their closest points, it computes for each cluster a set of $c$ "well-scattered" representative points, and then scaling them towards the mean of the cluster by a fixed factor; the linkage between two clusters is then the distance between their closest representative points.

A set of well-scattered representative points for a cluster can be chosen by first choosing the point furthest from the mean, then repeatedly choosing points furthest from the closest one of the previously chosen points. This requires scanning over the set of points $c$ times, giving a runtime of $O(n)$, which is undesirable since the set of representative points must be recomputed each time two clusters are merged. An alternative is to compute the $c$ representative points for a merged cluster from the $2c$ representative points from the two clusters it was formed from. This sacrifices quality of the representative points to give a $O(1)$ runtime, but not by much since the new points are chosen from points that were well-scattered over the previous clusters.

The effect of choosing representative points is to capture the shape of a cluster, while scaling them towards the mean limits the effects of outliers. This avoids the problem that single-link algorithms have with respect to chains of outliers connecting two otherwise disparate groups. By taking into account the shape of the cluster, it is possible to identify non-spherical shapes (such as elongated regions), which would not be possible if only the mean was considered. However, it still encounters problems if the shapes of the clusters are not convex.

Efficiently computing a clustering using this linkage metric uses a procedures similar to that described above for the single-link metric. However, instead of storing a matrix of distances and array of closest clusters, we now maintain a heap of clusters sorted by each cluster's distance to the closest neighboring cluster, and a *kd*-tree of all the representative points. With these data structures, identifying and merging a pair of clusters requires extracting the minimum element from the heap, then merging that cluster with its closest neighbor and generating the new set of representative elements. To update the data structures, the representative points for the pre-merge clusters are removed from and the new representative points added to the *kd*-tree. We then scan over all other clusters to determine which is closest to the new merged cluster, and to recompute the closest cluster (using the *kd*-tree) for all clusters that were previously closest to one of the merged clusters; we update the heap accordingly. Since each data structure operation requires $O(\log n)$ time, each merge operation requires $O(n \log n)$ time, and the algorithm's total runtime is $O\left(n^2 \log n\right)$. Note that this is slower than the SLINK algorithm, though for low-dimensional data sets CURE can be shown to run in the same $O\left(n^2\right)$ time bound.

CURE also employs the standard streaming technique of random sampling to handle large data sets. This introduces a tradeoff between speed and accuracy, but it is typically justifiable in light of the fact that most clusters contain enough points that, with high probability, every cluster will be sampled with a sufficiently large sample size. An argument based on Chernoff bounds makes this reasoning precise. Sampling also has the useful side-effect of reducing the number of outliers, as they are less likely to be sampled than points in clusters.

The results, as shown by an experimental comparison of CURE with other hierarchical algorithms including single-link and all-pairs [10], are that single-link clustering is best for handling non-spherically shaped clusters, but the data set must have clearly-defined clusters without outliers. CURE trades off some ability to detect arbitrarily-shaped clusters for robustness in the face of outliers, so is more suitable for general data sets. The results also suggest that sampling is a feasible technique for increasing scalability without overly degrading result quality.

### 2.3 Non-numeric and Categorical Data: ROCK

The algorithms described above are all intended to find clusters in sets of numeric data in metric spaces. Unfortunately, many data mining problems involve data that is boolean or categorical. The classic example for this is the *market basket problem*: examining a large database of supermarket transactions to determine clusters of items that are frequently purchased together. In this case, each purchase is a data point consisting of many Boolean variables, each indicating whether a certain item was included in that purchase.

Of course, this problem can still be expressed as a numerical clustering problem which has one dimension per item, where the data points take value 1 in a certain dimension if the corresponding transaction contains that item and zero otherwise. However, such data sets generally have other characteristics that make standard clustering algorithms produce poor results. For one, each transaction includes only a small subset of the (perhaps thousands) of total items. Thus, it is reasonable for a cluster to consist of a larger class of items that are frequently associated, whereas each individual transaction in the cluster contains only a small number of items in that class. The effect of this is that two items that should be in the same cluster may actually have no items in common, but rather be "connected" via other transactions that share many of the same items.

None of the single-link, all-pairs, or CURE algorithms work well for these types of data sets. The data set size is generally large enough that there will be occasional transactions that contain items that appear in different clusters, bridging the clusters and causing them to be merged by a single-link algorithm [11]. The all-pairs and (to a lesser extent) CURE algorithms use the cluster mean as a factor in determining the linkage of clusters. This leads to a "ripple effect" [9]: as a cluster becomes larger, it contains increasingly more points with more different items, and so the averaging process leads to a mean that contains a smaller fraction of a greater number of items. Then linkage metrics based on the mean essentially lose their information about what distinguishes the items in the cluster, and cannot tell the difference between data points that are largely similar but differ on one or two items and other averaged data points that differ on every item, but in small amounts.

The ROCK algorithm [11] is designed to handle data sets made up of Boolean attributes; it also handles categorical data by treating each categorical attribute as a set of Boolean attributes for each possible value it can take on. It achieves this goal by defining a different sort of linkage metric. As before, it assumes the existence of a distance function $d(p, q)$ — for the market basket example, often this is the Jaccard coefficient $1 - \frac{|I_p \cap I_q|}{|I_p \cup I_q|}$, i.e. the fraction of items in the two transactions that are not shared between them. It defines two data points to be *neighbors* if their distance is less than a user-specified parameter $\theta$, and defines $link(p, q)$ to be the number of common neighbors shared between two points $p$ and $q$. Then the linkage function between two sets is based on the number of cross-links between points in each set:

$$linkage(A, B) = \frac{(|A| + |B|)^{1+2f(\theta)} - |A|^{1+2f(\theta)} - |B|^{1+2f(\theta)}}{\sum_{\substack{p \in A \\ q \in B}} link(p, q)}$$

The denominator of this linkage function is the number of cross-links, a fairly intuitive measure of the linkage of two sets; the numerator requires a bit more explanation. Using the number of cross-links alone is not sufficient because a large cluster contains more points and thus more possibilities for "outlier" cross-links to other clusters; large clusters will then have a tendency to inappropriately consume smaller clusters. The numerator of the linkage function attempts to minimize this effect. The user-supplied function $f(\theta)$ is chosen so that each point in cluster $C$ is expected to have approximately $|C|^{f(\theta)}$ links to other points in the cluster. Thus the expected number of links between points in a cluster $C$ is $|C|^{1+2f(\theta)}$. So the numerator is the expected number of cross-links between the two sets: the expected number of links between the two sets together minus their expected numbers of links individually. The linkage function is lowest, and so two clusters are most closely linked, when they have many more cross-links than would be expected from their size alone.

Whereas the previous hierarchical algorithms make decisions "locally", considering only the distance between points in a cluster, ROCK uses a somewhat more global approach, taking into account information about their neighbors in the form of the link count. This provides improved result quality, but necessarily increases the runtime cost. A necessary first step is to count the number of links between each pair of data points, which can be accomplished by assembling every point's neighbors into an adjacency matrix then squaring it ($O\left(n^{2.37}\right)$ time using a Strassen-like algorithm). Alternatively, for the common case where each point has few neighbors, the simple approach of calculating for each point its set of neighbors then incrementing the link count for every pair of neighbors gives the same result with a runtime of $O\left(n m_a m_m\right)$, where $m_a$ and $m_m$ are the average and maximum number of neighbors per point.

Once the link counts are computed, an agglomerative algorithm largely similar to that described in Section 2.1. It requires a local heap for each cluster containing all other clusters it has links to, ordered by the linkage metric, and also a global heap containing all clusters ordered by their lowest linkage metric. Until the desired number of clusters has been reached, the minimum is extracted from the global heap, giving the two clusters to be merged. They are then merged and the link counts and heaps are updated. Note that the number of links between any cluster and the newly merged cluster is simply the sum of the number of links between it and each of the two pre-merge clusters, so updating the link count simply requires summing $O(n)$ entries. The local heap must be rebuilt and the global heap updated; this requires $O(n \log n)$ time, so the algorithm's total runtime is $O\left(n^2 \log n + n m_a m_m\right)$.

As with CURE, sampling is used to reduce the runtime of the algorithm while still giving a result of acceptable quality.

The ROCK algorithm provides much better handling for categorical data sets, a common class for which traditional clustering algorithms give poor results. Besides the increased runtime, a disadvantage is the increased number of input parameters. The user must specify not only the desired number of clusters $k$ but also the threshold $\theta$ for classifying a point as a neighbor and a function $f(\theta)$ that gives an estimate for the number of neighbors for a point. In practice, however, the authors claim that $f(\theta)$ needs not be a very good estimate [9].

# 3 Partitioning Algorithms

Whereas hierarchical algorithms incrementally build clusters either by agglomerating smaller clusters or dividing larger ones, partitioning clustering algorithms attempt to directly identify clusters by partitioning the entire data space.

## 3.1 $k$-means

We are obligated to begin with a mention of the $k$-means algorithm, since it finds wide use in statistical and scientific applications. As we shall see, however, it suffers from many of the problems previously described. While popular in practice, from an algorithmic standpoint it falls short.

The $k$-means approach is to represent each of the $k$ clusters by the mean (centroid) of the points contained within it. Then the goal is to minimize the sum of the distances from each point to the mean of the cluster that contains it. This has an intuitive geometric meaning, which is pleasing for numeric data, though it is not particularly useful for categorical data. However, it uses the mean, which makes the algorithm very sensitive to outliers since points that do not fall into any cluster can still skew the centroid of the nearest cluster. It also identifies only spherical clusters, which means it does not work will with data sets containing clusters of non-uniform shapes.

The classic algorithm for solving this problem is Forgy's algorithm [7], which begins by selecting the initial $k$ cluster centroid assignments at random. It then iteratively improves the clusters by assigning each point to the nearest centroid, then recomputing the centroid of each cluster; this process is repeated until no changes occur.

This is a hill-climbing algorithm, and suffers from the standard problem with such algorithms: it will find a local minimum, not a global minimum — clearly a major problem from a correctness standpoint. It is difficult to make any guarantees on the runtime, since the number of iterative improvements required is unknown; in fact, in some cases, the algorithm fails to converge entirely.

### 3.2 $k$-median: Approximations

A more useful variant on the $k$-means problem is the $k$-median problem. As before, the goal is to choose $k$ cluster centers so that the total distance from each data point to its nearest center is minimized. Now, however, the centers are required to be actual data points, rather than the mean of several data points. Note that this differs from the related $k$-centers problem in that both involve choosing $k$ points as centers, but $k$-median requires minimizing the total (or, equivalently, average) distance between each point and its center, whereas $k$-centers requires minimizing the *maximum* distance.

This is an improvement over $k$-means because it provides improved handling of outliers, in much the same way as the median of a set of numbers is preferable to the mean. It does, however, still assume that the clusters are spherical. Moreover, solving the problem is non-trivial since it is NP-complete. Thus we are forced to consider approximation algorithms.

Charikar et al. [4] were the first to identify a constant-factor approximation for the metric $k$-median problem. The approximation algorithm expresses $k$-median as an integer linear program, with one set of Boolean variables indicating for all $i$ whether point $i$ is a center and another set indicating for all pairs $\langle i,\ j\rangle$ whether point $j$ is assigned to the cluster centered at point $i$. The relaxation of this integer linear program allows "fractional centers," where the values of fractional centers still sums to $k$. They then round the solution to this linear program first into a $\{1/2, 1\}$-integral solution (which can have only full and half centers), and then into an integral solution, giving a $6\ 2/3$ approximation of the optimal solution. This algorithm, however, is of theoretical rather than practical importance.

An algorithm commonly used in practice for computing an approximation of the $k$-median clustering is PAM [12]. It is an iterative refinement algorithm similar to Forgy's algorithm for $k$-means: it selects $k$ cluster centers at random and assigns each point to the closest center. It then repeatedly considers replacing each cluster center with a different point in the cluster, and performs this reassignment if it improves the sum-of-distances objective function. As with Forgy's algorithm, this method has the major drawback that it may find a local minimum instead of the global minimum. The iterative refinement process also leads to a high runtime. CLARA [12] is a related algorithm that performs better on larger sets by taking several relatively small samples of the data set, applying PAM to each of them, and taking the best result.

## 4 Density-based Algorithms

Density based cluster algorithms are a special type of partitioning algorithms, so named because of the criteria they use to determine if a given point should be added to a cluster. While other algorithms typically just use the distance between two points to make such decisions, these algorithms use the density of data points around the point in question in making the choice to include.

### 4.1 DBScan

DBScan is one of the best-known density-based clustering algorithms. It takes two parameters, $\epsilon$ and *MinPts*. A point is part of a cluster if a circle of radius $\epsilon$ centered at that point contains at least *MinPts* data points. If this is not the case, then that point is instead treated as an outlier, and discarded. Two points $p_1$ and $p_2$ are *density-reachable* if the distance between them is less than $\epsilon$. Points $p_0$ and $p_n$ are *density-connected* if there exists a sequence of density-reachable points connecting the two. In this algorithm, a cluster is defined as a set of points that are all density-connected [5]. When DBScan runs, it expands outward from existing clusters, testing points close to each cluster to determine if they too are included in the cluster. Then, it examines other points to see if they are part of a new cluster, or if they are simply outliers.

DBScan has some major advantages over some of the clustering algorithms that preceded it. It is more effective at identifying arbitrarily sized and shaped clusters than most other algorithms, and is very successful in spatial clustering [20, 19]. This is, in fact, a general property of density-based algorithms. Unfortunately, for the same reason that it does well with more varied input shapes, it also has problems with chaining effects, much like SLINK (Section 2.1). If a bridge of outlier points connects two clusters, DBScan may be unable to distinguish the two clusters, and will sometimes produce an output in which they are combined into one large cluster. It is still more robust than single-link algorithms, however, since DBScan will eliminate outliers that are not surrounded by at least *MinPts* other points within distance $\epsilon$. [10]

While DBScan in its simplest form works best with spatial clustering, it can be applied to other types of data sets as well [6]. In the ordinary form of DBScan, the condition used to determine if two points are neighbors is simply the inequality $d(p_1, p_2) < \epsilon$. For other types of data sets, we can use some other binary, symmetric, and reflexive function to judge whether two data points are neighbors or not. The rest of the algorithm remains the same. The result of this change is that now DBScan can be applied to higher-dimension data, or even to non-numerical data [6].

As with many of the other parameterized algorithms, choosing appropriate values for $\epsilon$ and *MinPts* can be difficult, and having good results depends heavily on having those appropriate values. DBScan will have particular trouble finding good values for its parameters if the different clusters vary greatly in density. The choice of parameters can have fairly dramatic effects on the clustering that is produced, ranging from no clusters at all to one large cluster containing all the points in the data set [20].

### 4.2 OPTICS

OPTICS is a hierarchical variation of DBScan which attempts to alleviate some of its predecessor's weaknesses [2]. In particular, DBScan has trouble dealing with data in which different clusters have very different densities, because it is hard to select a value for $\epsilon$ that will work for all of them. OPTICS avoids this issue by using density-based cluster ordering. It orders partial clusters by their densities, beginning with the most dense. If a particular set of data points is a cluster when $\epsilon$ is very small, that cluster has a high density. If $\epsilon$ is increased, the size of the cluster will likely also increase, but it will still contain the original smaller cluster.

OPTICS gathers and stores data that allows the algorithm to recognize clusters of varying densities when the final consolidation scan occurs. These modifications allow OPTICS to produce more accurate outputs than DBScan can. It can also be used to analyze the intrinsic clustering structure of the data input to it. However, due to the overhead from storage and extra computation OPTICS is actually less efficient than DBSCAN. The long runtimes associated with it mean that OPTICS cannot be used for high-dimensional data sets, or for databases that are extremely large. There have been some attempts to improve on OPTICS' run time, but as is to be expected, such attempts require that some accuracy be sacrificed.

# 5 Grid-based Algorithms

While density-based algorithms are so called because of the criteria they use for deciding if a point should be included in a cluster, grid-based algorithms are named for the way in which they split up the data to be processed. These algorithms use a grid to divide the space that the data points are in, and then perform calculations on individual cells of the grid. There are generally fewer cells in the grid than there are data points in the input, and calculations are performed on the cells rather than on the individual points. This decreases the amount of objects on which calculations need to be performed, making the algorithm more efficient.

Both of the grid-based algorithms we will discuss here are also density-based to some extent. That is, after the algorithm applies the grid to the space in question, the density of points in a given cell determines whether it is in a cluster or not. To explain this concept further, we will first consider the CLIQUE algorithm.

## 5.1 CLIQUE

CLIQUE [1] is both density and grid based. It first partitions the space into a grid of non-overlapping rectangular cells by partitioning each dimension at equal length intervals. For each cell, the algorithm calculates the number of points in that cell, and if that number is greater than a given fraction of the total number of points, then that cell is *dense*, and thus considered to be part of a cluster. The threshold fraction is a parameter to the algorithm, as is the number of intervals that a dimension gets divided into.

CLIQUE first finds and sorts all subspaces that are above the threshold by their densities. Adjacent subspaces are combined until a whole cluster is contained in the union of those spaces, and then parts of that subspace that are not dense are pruned. When the pruning is done, that subspace is defined to be a cluster, and is expressed in the output as a disjunctive normal form (DNF) expression. The DNF output form allows the algorithm to describe clusters that exist in multiple subspaces. That is, there are multiple clusters, each representing one relationship, and they overlap to form a multi-dimensional cluster. The DNF output can be interpreted, showing to some extent what relationships were involved in the formation of the cluster.

CLIQUE's run time is exponential in the highest dimensionality of any cluster in the data set, but empirically it scales linearly with input, and scales well with added dimensions. It does particularly well with sparsely populated, high-dimensional data. The algorithm's output is independent of the order in which data was input. The accuracy of CLIQUE's output can sometimes be compromised by the simplicity of the algorithm. Also, the threshold density value is a fixed number, so CLIQUE can sometimes have trouble with varied densities across clusters. There have been some tweaks made on the original algorithm to try to improve it, such as the MAFIA algorithm, but in the interests of space, these will not be discussed here.

### 5.2 WaveCluster

While WaveCluster [17] is another grid-based and density-based clustering algorithm, its workings are, in fact, quite different from those of CLIQUE. In particular, WaveCluster is unique from anything discussed thus far in its use of wavelets for calculation. The algorithms parameters are the wavelet to be used, the number of transformations $k$ to be performed, and the number of grid cells that each dimension will be divided into. As with CLIQUE, the space is first divided into equal-volume non-overlapping cells. Then the given wavelet transform is applied to the cells, producing a new space, also divided into cells. In this new space, we find densely populated cells, and any maximal group of densely populated cells is a proposed cluster. This is repeated $k$ times.

Wavelets are primarily used in signal processing. They contain a low-pass filter, which helps to remove outliers. Though a full discussion of the theory of wavelets is of course beyond the scope this paper, the nature of wavelets is to suppress weak data and enhance stronger data, so applying a wavelet to the input space will make the natural clusters become more apparent. Wavelet transformation is fast, which allows the total run time to remain linear in the input size. Also, the multiple applications of the wavelet allow the algorithm to find clusters of varying densities, as each transformation will make increasingly fine clusters stand out.

WaveCluster's output is independent of the order in which the data points were input. It is not sensitive to noise, and can detect clusters of arbitrary shape. It performs best in lower dimensions, where there are more data points than there are cells in the grid. This gives a linear runtime bound. However, since the number of cells in the grid is exponential in the number of dimensions, for higher dimensions the size of the input might not be larger, and the linear bound no longer holds.

## 6 Conclusion

In this paper, we have reviewed algorithms from the following families: hierarchical, partitioning, density-based, and grid-based. Partitioning algorithms are very popular in practice, but they have the downsides of long run times and the possibility of returning local minima instead of global ones. Also, they have difficulty dealing with outliers and non-spherical clusters. Other algorithms, such as OPTICS, also have fairly inefficient run times, particularly when faced with high-dimensional data. However, this sacrifice in efficiency is made in the interest of more accurate results when clusters are arbitrarily shaped.

Many of these algorithms are parameterized, which introduces the difficulty of determining appropriate values for those parameters. For example, DBScan can perform quite well on abitrarily shaped clusters, but only if the parameters it has are well-suited to the densities of the clusters in the data. While most of these algorithms are meant for numerical data, Boolean data can be clustered as well, as demonstrated by ROCK. CURE can deal with elongated clusters, but not completely arbitrarily shaped ones. In a more extreme case, DBscan and WaveCluster work well with arbitrary shapes.

There are many other clustering algorithms that are not included here, but these were chosen because they give a good cross-section of the clustering options that are currently available. Each of the algorithms we have discussed here has its own strengths and weaknesses, and while they all aim to solve the same problem, they vary greatly in implementation. There is no particular algorithm here that is the best across the board. When someone is in need of a clustering algorithm, they should consider all of these options to determine which type of algorithm would work best for their particular situation.

## References


[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. Int. Conf. on Management of Data (ACM SIGMOD '98)*, pages 94–105, 1998.

[2] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *Proc. Int. Conf. on Management of Data (ACM SIGMOD '99)*, Philadelphia, PA, 1999.

[3] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[4] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k -median problem (extended abstract). In *ACM Symposium on Theory of Computing*, pages 1–10, 1999.

[5] DBScan clustering algorithm. `http://www.inf.unibz.it/dis/teaching/msc/project-dbscan.html`.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery, An International Journal*, 2(2):169–194, June 1998.

[7] E. Forgy. Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications. *Biometrics*, 21(3):768, 1965.

[8] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice, 2003.

[9] S. Guha, R. Rastogi, and K. Shim. ROCK: A robust clustering algorithm for categorical attributes. Technical report, Bell Laboratories, Murray Hill, NJ, 1997.

[10] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of ACM SIGMOD '98*, Seattle, WA, 1998.

[11] S. Guha, R. Rastogi, and K. Shim. ROCK: A robust clustering algorithm for categorical attributes. In *Proceedings of the 15th International Conference on Data Engineering*, pages 512–521, Sydney, Australia, 1999. IEEE.

[12] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data*. Wiley, March 1990.

[13] H.-P. Kriegel, P. Kroger, and I. Gotlibovich. Incremental OPTICS: Efficient computation of updates in a hierarchical cluster ordering. In *Proc. 5th Intl. Conf. on Data Warehousing and Knowledge Discovery (DaWaK '03)*, pages 224–233, Prague, Czech Republic, 2003.

[14] K. Liu, D. Zhou, and X. Zhou. Clustering by ordering density-based subspaces. In *Proceedings of the Sixth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'02)*, pages 28–39, Taipei, Taiwan, May 2002. Springer-Verlag.

[15] C. F. Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 21:1313–1325, 1995.

[16] L. Parsons, E. Haque, and H. Liu. Evaluating subspace clustering algorithms. In *SIAM International Conference on Data Mining 2004*, Lake Buena Vista, Florida, April 2004. SIAM.

[17] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 428–439, 24–27 1998.

[18] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[19] C. tien Lu. Spatial clustering methods in data mining. `http://www.cs.umn.edu/research/shashi-group/paper_ps/Spatial-Clustering%.ps`.

[20] O. R. Zaiane, A. Foss, C.-H. Lee, and W. Wang. On data clustering analysis: Scalability, constraints and validation. In *Proceedings of the Sixth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'02)*, pages 28–39, Taipei, Taiwan, May 2002. Springer-Verlag.