Clusterers: a Comparison of Partitioning and Density-Based Algorithms and a Discussion of Optimisations

David Breitkreutz
School of Mathematics, Physics & Information Technology
James Cook University
Townsville, QLD 4811, Australia
David.Breitkreutz@jcu.edu.au

Kate Casey
School of Mathematics, Physics & Information Technology
James Cook University
Townsville, QLD 4811, Australia
Kate.Casey@jcu.edu.au

Abstract

Though data mining is a relatively recent innovation, the improvements it offers over traditional data analysis have seen the field expand rapidly. Given the critical requirement for the efficient and accurate delivery of useful information in today's data-rich climate, significant research in the topic continues.

Clustering is one of the fundamental techniques adopted by data mining tools across a range of applications. It provides several algorithms that can assess large data sets based on specific parameters and group related data points.

This paper compares two widely used clustering algorithms, K-Medoids and Density-Based Spatial Clustering of Applications with Noise (DBSCAN), against other well-known techniques. The initial testing conducted on each technique utilises the standard implementation of each algorithm. Further experimental work proposes and tests potential improvements to these methods, and presents the UltraK-Medoids and UltraDBScan algorithms. Various key applications of clustering methods are detailed, and several areas of future work have been suggested.

1. Introduction

Traditional statistical analysis relies on confirmatory methods; applying pre-defined mathematical models to data sets in order to identify trends. This technique, while effective and extensively used, has limitations when presented with very large data sets. Data mining, however, is based on an exploratory framework and analyses data on an adhoc basis. This provides additional flexibility to the suite of algorithms, and allows efficient processing of large reserves of data.

As a subsidiary of data mining, clustering is promoted as an extremely powerful means of grouping related data points. The technique can efficiently reveal highly relevant trends in a source data set, and this capability extends to the large data repositories used by scientists, researchers, and businesses.

As a result, the field has developed into one of the foremost research areas in modern computing. There exist numerous algorithms for performing such analysis, and each may be more suitable to certain circumstances, depending entirely upon domain-specific parameters. As previously discussed, these techniques can analyse both large-scale databases and data warehouses.

The efficiency and accuracy of results from these data mining tasks relies directly upon the choice of a suitable algorithm. Thus, given the many different types of data sets that exist, there is a strong requirement for further research into the improvements of these techniques.

1.1. Applications

Data mining in general has a multitude of applications across a wide variety of fields. Pattern recognition for image analysis, medical diagnostics, sales forecasting and weather prediction are a recognised as a few of the more traditional usages. However, due to extensive development in the field and the recent explosion in data recording, the capabilities extend far beyond these basic functions. On-board computer analysis in vehicles, product quality analysis, targeted advertising campaigns, spam email filtration, fraud detection, and online crime analysis are but a few of the fields into which data mining now extends [14]. Clustering applies to all of the aforementioned applications as a subset of data mining.

A prominent example of this functionality is the application of emergency situation data analysis; for example, data recorded by authorities during forest fires. This paper will

investigate that specific scenario in section 5.1, with the test sets utilised containing fire data from Portugal.

1.2. Other methods

There are a wide variety of clustering methodologies that exist within this field. These include, but are not limited to, grid-based categorisation, density-based grouping, hierarchical segmentation, and constraint-based analysis. As these techniques have been already widely researched, there are many existing works that compare and contrast each. The interested reader is directed to [9] for an expansive summary of these topics.

The research outlined in this paper is concerned with the application of two of the most widely used methods; *partitioning-based* and *density-based* clustering.

Partitioning algorithms are effective for mining data sets when computation of a clustering tree, or *dendrogram*, representation is infeasible [13]. In particular, K-Medoids is highly efficient for smaller data sets with spherical-type clusters. However, due to the inherent swapping of medoids to optimise the clustered solution, the algorithm suffers greatly as large data sets are introduced.

Density-based algorithms perform optimally when operating upon spatially-indexed data. The methods provide benefits when analysing data sets that contain high levels of noise or when clusters are arbitrarily shaped. Specifically, DBSCAN is able to grow clusters within a specified neighbourhood whilst the minimum points threshold is not satisfied, thus efficiently dividing real data and noise in a variety of shapes.

1.3. Purpose of paper

This paper focuses upon the DBSCAN and K-Medoids algorithms from the density-based and partitioning families respectively. It outlines the design, advantages and disadvantages of each method, and studies similar methodologies. The work contained goes on to propose both basic implementations of the techniques, as well as initiating new research into improving the efficiency of the aforementioned methods. A great deal of literature already exists regarding these techniques, and a review of these documents can be found in Section 2.

1.4. Experimental data set

The data set chosen for the comparison of and amendments to the K-Medoids and DBSCAN techniques is *forest fires* [5]. The set contains environmental data from fire instances in northeastern Portugal; representing instances within space, and including attributes such as date, temperature, humidity, wind conditions, rain information and burned area.

Analysing this data set is necessary as forest fires cause millions of dollars worth of damage to property and claim many lives each year. If clustering techniques can be used to more accurately determine the patterns of such fires given prevailing environmental conditions, scientists and researchers will be able to achieve further understanding of the phenomenon. This knowledge will reap enormous benefits; predictive techniques can be updated, analysis centres local to fires can more accurately forecast fire behaviour, government agencies can be continuously informed of these conclusions, and residents and business owners can be warned earlier. This improved system will result in reduced property damage, lower restoration costs and, ultimately, fewer lives lost.

1.5. Results overview

The results explained in section 5 form the basis of the experimental portion of this paper. Based on the forest fires data set, these figures reflect the implementation results of the DBSCAN and K-Medoids algorithms.

In addition, testing of the modified algorithms this paper suggests has shown a level of computational improvement for both methods when clustering the aforementioned data set. These findings are detailed and discussed in section 5.

2. Related Work

2.1. Data analysis

A formal definition by Moore [20] states that data analysis is *"the examination of data for interesting patterns and striking deviations from those patterns"*. Such processing can lead to conclusions which assist a multitude of people in varying tasks; from medical diagnostics to population growth predictions to forest fire forecasting. The potential developments in these research areas due to quality data analysis are not only beneficial in terms of further knowledge and understanding, but will inevitably benefit ordinary people in ways beyond the scope of this paper. Data mining is a subset of the broader data analysis field, and as such can provide the advantages previously discussed when large data sets require processing.

2.2. Clustering

Aggarwal et al. [1] define the process of clustering as: *"Given a set of points in multidimensional space, find a partition of the points into clusters so that the points within each cluster are close to one another"*. Proximity is measured using a variety of algorithm-specific metrics, such that the closer two arbitrary points are to one another, the

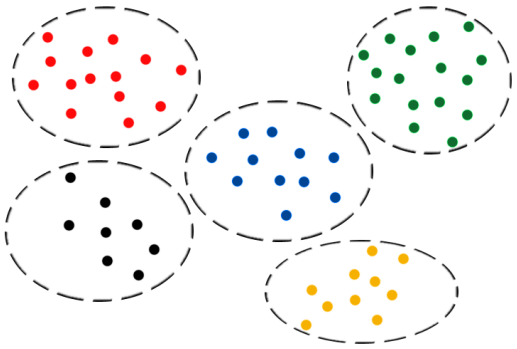

Figure 1: Example of clustered points

more strongly they are considered to be related. This process results in defined groupings of similar points, where strong inter-cluster and weak intra-cluster relationships exist among and between points, an example of which is detailed in Figure 1.

Clustering can be categorised in machine learning terms as a form of *unsupervised learning*; that is, clusters are representative of hidden patterns in the source data set[3]. Raw information is analysed and relationships are discovered by the algorithm without external direction or interference; learning through observation rather than by the study of examples[9]. In addition, this objectivity translates to an effective means of data analysis without the opportunity for subjective human conclusions to be drawn from data.

2.3. Partitioning clustering algorithms

Partitioning methods define clusters by grouping data points into k partitions, defined by the user at the time the process is executed. A point is determined to be *similar* to other points within its partition, and *dissimilar* to points that lie outside the boundary of that partition [9]. Comparison is based on the characteristics of the data set provided. Thus, the algorithms rely on the conversion of semantic data attributes (width, height, shape, colour, cost or others) into points that determine physical location on a set of mathematical axes. This provides an objective and computationally acceptable framework for analysis. In the simplest case only two attributes exist, and thus the conversion renders a point on a standard Cartesian plane. This process is greatly complicated when, as often occurs in highly detailed source sets, hundreds of attributes are present. The rendering plane takes on high dimensionality, and the complexity of analysis becomes very computationally expensive.

Partitioning offers several features, including efficiency in processing spherical clusters in small to medium data sets, and scalability across smaller applications. However, the algorithms suffer from extreme complexity and computational cost at higher dimensions in large sets, require the user to specify the number of clusters to be determined (k) and experience problems defining arbitrarily shaped clusters.

Previous work has outlined several clustering methods within the partitioning family; *K-means* converts data to numerical points on a set of axes and calculates the numerical mean of these points to dictate cluster centroids [10], *K-modes* is similar to K-means though it calculates the mode of point groupings instead of the mean to define cluster centroids , and *K-medoids* converts data points to objects on a set of axes and arbitrarily chooses different objects to act as cluster centroids until the clusters are optimised [17]. As K-Medoids provides dramatic improvements in noise immunity over its other partitioning counterparts, a great deal of further research has been pursued including the forthcoming work in this paper. *Partitioning Around Medoids* (PAM) randomly chooses medoids, then randomly calculates the result of swapping these with other random objects in an attempt to improve the clustering outcome [18]. This approach works adequately for small source sets, but does not scale well as complexity at each iteration is quadratic. *Clustering LARge Applications* (CLARA) takes a single random selection of points in the set as representative data and then applies PAM to this smaller subset [16]. Reducing the size of the set for analysis improves scalability as complexity at each iteration is linear, but the method suffers depending upon the quality of sample data chosen. *Clustering Large Applications based on RANdomised Search (*CLARANS) takes a random dynamic selection of data at each step of process – thus the same sample set is not used throughout the clustering process [21]. As a result, better randomisation of source data is achieved, but the method still suffers depending on the random selection and can be very slow as the algorithm's time complexity is approximately quadratic. *UltraK-Medoids*, the algorithm proposed in this paper, makes improvements to the basic K-Medoids method by performing reassignment of medoids and not assessing the effect on the data space through recalculation. This provides performance advantages, reducing time complexity to linear, but suffers when data sets are vastly dispersed.

2.4. Density-based clustering algorithms

The density-based group of clustering algorithms represent a data set in the same manner as partitioning methods; converting an instance to a point using the data attributes of the source set. The plane contains clusters with high internal density and low external density in a similar manner to its partitioning ancestor [19]. The process of adding points to a cluster is iterative, unlike partitioning methods. Nearest neighbours of each point can thus be investigated, arbitrary shapes formed, and existing clusters merged as the algorithm moves through all points. As a result, analysis

can easily isolate noise instances from relevant data, whilst being able to cluster data object sets that include hollow formations.

Density-based algorithms provide advantages over other methods through their noise handling capabilities and ability to determine clusters with arbitrary shapes (eg. a hollow circle, torus or other non-convex formation). As with partitioning techniques, computational cost is a disadvantage when the technique is used with large amounts of source data and sets containing excessive noise.

Literature exists on several techniques that utilise the density-based clustering concept. *DENsity based CLUstEring* (DENCLUE) uses multiple clustering paradigms (including density-based, partitioning and hierarchical), is based on a set of density distribution functions, and uses influence functions between points to model the data space [12]. The method has a strong mathematical foundation which provides advantages in terms of set representation, but density and noise parameter selection can adversely affect the average linear time complexity. *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) locates and grows regions with sufficient density as specified by input parameters [6]. The algorithm does not require that all points are allocated to clusters which provides tolerance to noise and supports clustering of arbitrary shapes, but can suffer if input parameters are selected incorrectly. Time complexity is typically $O(n \log n)$ but can be as poor as quadratic when a spatial index is not used. DBSCAN is widely used due to its ability to handle noise and define clusters with non-standard shapes, and thus many extensions to its algorithm are defined in the literature. *Ordering Points To Identify the Clustering Structure* (OPTICS) is similar to DBSCAN but produces augmented cluster ordering instead of defining actual clusters[2]. This approach is advantageous as the ordering can be used to both derive key cluster characteristics and analyse the structure of the cluster space, and time complexity is the same as DBSCAN; averaging $O(n \log n)$. *Generalized Density-Based Spatial Clustering of Applications with Noise* (GDBSCAN) can cluster both polygons and points, and does not rely on any set definition of a neighbourhood; the method can use non-spatial attributes rather than number of objects to define neighbourhood density *[22]*. The technique does not limit clusters to a maximum radius and thus achieves independence from a neighbourhood size input variable. Time complexity is similar to DBSCAN, though $O(n \log n)$ is achieved only using RTrees [4]. *UltraDBScan*, the algorithm proposed in this paper, introduces the concept of a dynamic cluster radius specification. Changing this value based on the number of points in the cluster as the algorithm moves through all its points allows dynamic removal of noise. This is highly advantageous in sets with closely-bound clusters, but may result in the declaration of valid points as noise in more dispersed sets.

2.5. Alternative clustering methods

Two of the most popular clustering methods in use other than density-based and partitioning are *hierarchical* and *grid-based*.

Hierarchical algorithms can define clusters in two ways; either top-down (*divisive*) or bottom-up (*agglomerate*). Divisive techniques define a broader cluster first which encompasses all points, then split this into more specialised child clusters. These child clusters in turn spawn their own child clusters and the process continues until all points are accounted for. The agglomerate technique is the reverse, defining the smallest and lowest-level clusters first and working upward until the entire set is contained in one cluster [8]. The methods are advantageous in terms of processing time efficiency, but accuracy suffers as they do not allow for incorrect clustering decisions to be reversed (back-traversal). Chameleon [15] and BIRCH [25] are examples of hierarchical methods; Chameleon concerns analysis of inter-object relationships at each hierarchical level and BIRCH utilises iterative relocation.

Grid-based techniques divide the data space into a finite number of cells (hence *grid*) and clustering methods are applied to these segments. Due to this division of study space, such algorithms are dependent in terms of processing only on the quantity of cells at any dimension. Thus the technique has very high computational processing efficiency [9]. The WaveCluster [23] and STING [24] algorithms both employ grid-based clustering techniques; WaveCluster utilises wavelet transformations on source data and STING exhibits typical grid-based analysis behaviour.

2.6. Alternative data mining techniques

Clustering differs from other classical data mining methodologies in a variety of ways and extensive literature already exists explaining this. For the purpose of conciseness, this paper will compare and contrast clustering only with the techniques of *classification* and *prediction*.

Classification is the process of constructing a learning model, or *classifier*, that assigns points in a data set to predefined classes[7]. Whilst this definition initially appears similar to that of clustering in that points are grouped based on attributes, any further similarities are thus exhausted. The technique utilises *supervised learning* as opposed to the unsupervised learning environment of clustering. The model is compiled by studying provided data (known as a training set) containing class labels; hence the technique is "supervised" due to example data exposure. Once the model is complete it can be applied to real, labeled data and assign each new tuple to its respective class[9]. Clas-

sification is used in the categorisation of discrete or unordered data sets; for example, determining the level of risk (high/medium/low) associated with a certain financial investment given pre-existing market data.

Prediction is similar to classification, in that the process of learning from existing data is supervised, and a model is constructed. However, prediction techniques are used when assessing continuous data sets [9]; for example, determining the price of a stock market share at some point in the future given previous performance trends. Unlike prediction, clustering is not exclusively used for forecasting future values, rather grouping like points for immediate assessment.

Whether to utilise clustering in analysis is a decision based on the purpose of the data mining task. If future predictions are required, an alternative technique should be employed, but if like points in the set are to be grouped, clustering is an appropriate choice.

3. Comparison

This paper aims to draw a detailed comparison between two general clustering paradigms - density-based and partitioning - and in order to do so, examines the differences between several other techniques. By examining the differences in how each method operates, their varying capabilities and the circumstances in which they are most applicable will be revealed.

3.1. DBSCAN

The DBSCAN clustering technique, first proposed in 1996 by Ester et al. [6], requires the user to specify two parameters: *minPoints*, defined as the minimum number of points required to exist in a neighbourhood to be declared a cluster, and ε, defined as the radius of the neighbourhood of a point based on a distance metric. The minimum number of points is used to determine if an arbitrary collection of points should be considered a cluster. The ε value controls the radius around an object, whether it be in Euclidean, Manhattan or Minkowski distance, in which the algorithm should consider other points as neighbours.

The process, as detailed in Algorithm 1, begins by considering an arbitrary data object that has not already been assigned to a cluster. The neighbours of this data point P are then located, according to the user-specified ε value. If the number of neighbours found is greater than the specified *minPoints* value, then the current data object should be added to the present cluster, and all neighbours should be processed recursively. Otherwise, if the number of nearby points is less than *minPoints*, then the data object is considered noise. This process then continues for all of the neighbours, processing them accordingly. At such time that a given cluster has been fully explored, the process begins again, until all points have been visited. At its conclusion, all defined clusters are finalised; no further points can be added to any cluster and any unclustered points are declared as *noise* [9].

Algorithm 1 DBSCAN

1. For each unvisited point P within the data set D
   (a) Get the neighbours of the P, according to the given epsilon distance
   (b) If the number of neighbours is equal or greater than the user-specified cluster threshold
      i. Increment cluster identifier counter
      ii. Add the given point to the current cluster
      iii. Recursively process all neighbours
   (c) Else if the point has fewer neighbours than the threshold value,
      i. Mark P as noise

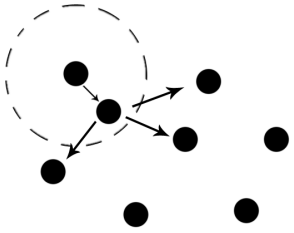

Figure 2: Initial stage of DBSCAN technique

An initial processing stage for the DBSCAN procedure, performed on a simplified data set, is shown in Figure 2. The dashed line represents a two-dimensional ε distance radius and the minimum points threshold is 1.

3.2. OPTICS

The Ordering Points to Identify the Clustering Structure (OPTICS) algorithm is procedurally identical to that of the previously mentioned DBSCAN [2]. Thus its algorithm is similar to that shown in Algorithm 1, and its time complexity is the same.

The OPTICS technique builds upon DBSCAN by introducing values that are stored with each data object; an attempt to overcome the necessity to supply different input parameters. Specifically, these are referred to as the *core-distance*, the smallest epsilon value that makes a data object a core object, and the *reachability-distance,* which is a measure of distance between a given object and another. The *reachability-distance* is calculated as the greater of either the *core-distance* of the data object or the Euclidean

Algorithm 2 K-Medoids

1. Arbitrarily select k objects as initial medoid points from given data set
2. Whilst swaps are occurring
   (a) Associate each data object in the data set with its closest medoid
   (b) Randomly select a non-medoid object, O
   (c) Compute the total cost of swapping initial medoid object with O
   (d) If the total cost of swapping is less than the total cost of the current system then
      i. Swap the initial medoid with O
   (e) Continue until there is no change

distance between the data object and another point[9].

These newly introduced distances are used to order the objects within the data set. Clusters are defined based upon the reachability information and core distances associated with each object; potentially revealing more relevant information about the attributes of each cluster.

3.3. K-Medoids

The K-Medoids clustering technique is concerned with the concept of partitioning data objects into similar groups. The technique was first developed in 1987 by Kaufman and Rousseeuw [17] and seeks to reduce the impact of noisy data experienced in simpler partitioning algorithms, such as K-Means. Instead of converting data attributes to simple points, it retains all the information from each record and represents the points as objects on a detailed axes.

This method, as detailed in Algorithm 2, segments the given database of objects into a user-specified number of clusters (k). Each cluster exists such that all contained data objects are most closely associated with the medoid of the cluster. The process begins with the initial selection of k arbitrary points as representative medoids, and then computes cluster assignments for each of the data objects. At this stage, each point will be associated with a medoid from the k medoids available, as demonstrated in Figure 3. In this example, the two marked data objects are considered to be medoids of their respective clusters. During this initial stage of the algorithm, the surrounding objects are assigned to their closest respective medoid.

The algorithm then continues by selecting a random non-medoid data object, O_r, and recalculating the potential benefit were it swapped with its current medoid. If there is such a benefit, the system should make the change and continue. This process continues until the system is in such a state that every data object has been assigned to a cluster, and further swapping would provide no further optimisation to the clustering outcome [3].

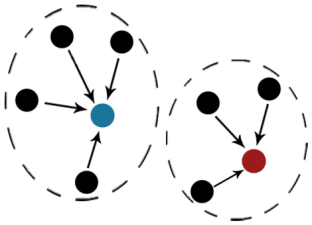

Figure 3: K-Medoids clustering process

Algorithm 3 K-Means

1. Arbitrarily or heuristically select k objects as initial centroids for the given data set
2. Whilst centroid changes are occurring
   (a) Associate each data point P in the data set to its closest centroid
   (b) For each centroid C,
      i. Recompute location according to data points associated with it
   (c) Continue until stopping criteria met

The calculation of prospective benefit after swapping is based on a cost function; the total distance between all points and their respective medoids. In a similar manner to DBSCAN, the term *distance* used in this case may refer, depending upon the implementation, to a Euclidean, Manhattan or Minkowski metric.

3.4. K-Means

The K-Means technique is similar to that of K-Medoids in that both methods use the same designations of input variables, and both attempt to cluster data objects in a similar manner. Whilst K-Medoids is concerned with using data objects themselves as cluster mid-points, K-Means, utilises the centroid, or *mean*, of a cluster for its computations.

The method, as described in Algorithm 3, begins as the K-Medoids method does - by using some process to select k-many objects as initial mean objects from the data set. From this, each data object is iteratively assigned to its respective cluster, according to its nearest centroid point. Once this process has completed, the centroid for each cluster is recomputed accordingly. This process will continue

until the data set has reached a state of convergence; some stopping condition has been reached, and the clustering results have thereby been sufficiently optimised [11].

3.5. Capability comparison

Directing responsibility for the choice of correct input parameters toward the user has clear disadvantages. Given different circumstances in terms of data shapes and data set size, markedly different results can be seen using the same algorithm. K-Means, K-Medoids and UltraK-Medoids require the user to nominate a desired number of clusters, and an incorrect choice can seriously compromise results. If too high a number is chosen, related points may be clustered apart, whilst too low a choice may result in noise points being allocated to a legitimate cluster. DBSCAN and OPTICS support location of an arbitrary number of clusters (no *k*-cluster limitation), but they require both a cluster radius and minimum-points-per-cluster specification, and thus suffer similar problems in terms of cluster accuracy to the K-algorithms. Incorrect choice of these variables can lead to effects on the final clusters similar to those experienced when using partitioning techniques. UltraDBScan, however, alleviates this issue and improves noise removal through allowing dynamic change of ε by the algorithm during processing. The minimum points variable is still required, though the effect of a pre-defined ε is reduced. This is highly effective when analysing tightly-grouped sets, but may lead to relevant relationships being discarded in less compact sets.

K-algorithms require that all points in the set are clustered, whereas DBSCAN and its descendants do not. This disadvantages the K-series as even noise points are required to be assigned to clusters, which compromises the quality of legitimate clusters. DBSCAN-type algorithms are not susceptible to this issue due to their design, and will correctly discard such points as noise.

The DBSCAN algorithm, like other density-based techniques, is most suited to data sets where the number, shape, or size of clusters present are not known, and the presence of noise would otherwise severely affect results. These are two benefits which are not available from the partitioning techniques - K-Medoids or K-Means - under review. Both of these algorithms are sensitive to noise, though K-medoids has slightly improved outlier tolerance than K-Means.

K-Means will scale more effectively than K-medoids due to its reduced processing requirements, however density-based algorithms are far more applicable to larger data sets due to their inherently robust nature. The K-algorithms work very well for smaller data sets (in terms of both dimensionality and quantity of records), but lose efficiency as the size of the source set increases. In terms of efficiency, which is directly related to scalability, K-algorithms perform better than density-based for small sets, but are surpassed in performance by DBSCAN and OPTICS when larger sets are introduced. This conclusion is supported by the time-complexity metric for each technique; K-algorithms operate in quadratic time, whereas density-based have $O(n \log n)$ average times, and quadratic only in worst-case scenarios.

Given the aforementioned observations and conclusions, traditional K-algorithms are recommended for low-dimensional data sets with few records, low noise and spherical data groupings. The original density-based algorithms are well-equipped to handle more expansive sets with higher dimensionality, larger quantities of tuples, widespread noise and arbitrarily-shaped cluster formations. Of the adapted algorithms proposed in this paper, UltraK-Medoids is recommended for the same style of data set as its ancestors, while UltraDBScan is recommended for lower-noise sets of the type suited to its ancestors.

4. Customisations

As previously mentioned, the subject area under consideration has already been extensively investigated by other authors over time. Many previous works have discussed ways in which clustering algorithms can be optimised and made otherwise more resilient. This paper attempts to add to the already voluminous amount of work in the field of clustering by introducing two new techniques. These new methods are derived from the existing DBSCAN and K-Medoids algorithms, which were chosen for analysis as they both operate successfully, return useful results, and are well established and understood throughout the existing literature.

It should be noted that these customisations aim only to modify the aforementioned techniques, rather than to construct extended methodologies. This choice is justified by the fact that any significantly complex modification to an existing clustering technique would affect its speed and performance, whilst leaving core functionality the same. As this paper focuses upon performance as a key aspect of clustering algorithms, research beyond the scope of the work contained herein would be required to investigate further performance enhancements. Thus, whilst this paper does not suggest entirely new clustering techniques, the modifications proposed do appear to have significant respective uses.

4.1. DBSCAN

Some authors [9] highlight that an issue which affects DBSCAN, and many other clustering techniques, is the necessity to correctly select input parameters. In the case of DBSCAN, these values are the minimum number of points needed to form a cluster, *minPoints,* and the distance value,

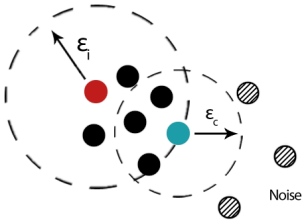

Figure 4: Customised DBSCAN procedure

ε 4.1. As expected, if a set of input variables are incorrectly selected - whereby the margin between correct and incorrect may vary greatly, depending on the data set - the results produced may be completely unexpected, and in some cases rendered useless.

This paper suggests that the DBSCAN algorithm be modified to offer dynamic change in ε distance, rather than using a static, potentially unsuitable value for all data objects within a collection. To achieve this change, the distance value is adjusted according to the number of points present within the current cluster. This modification aims to decrease the distance checked around each point within a cluster, in order to reduce the risk of data instances being included in a cluster to which they do not belong. Consider a dense cluster of objects, such as that in Figure 4, and that this cluster also has a number of outlier objects surrounding it. Using a constant distance value in this situation would be ineffective and result in the objects marked as noise being assigned to the cluster incorrectly. The consideration in these situations is that a very dense cluster will see its members similarly spaced. For any given ε, if many data objects are found using such a value, ε can be decreased to reduce the effect of noise, and lessen the potential for the *bridging effect*, whereby two clusters are inadvertently joined.

4.2. K-Medoids

As previously mentioned in Section 3.3, one of the biggest issues that affects the K-Medoids algorithm is the computational time required to determine the suitability of a given random object becoming a new medoid [9]. This overhead is largely unnecessary, as a system will frequently be in such a state that the reassignment of a given medoid will not affect a large amount of the data objects present. This would be especially relevant for a large data set that consists of many clusters; a single medoid change would only affect the small number of objects and clusters nearby.

Within the original algorithm, detailed in Algorithm 2, the step described at 2c has been determined as the most superfluous. At present, calculating the cost of swapping one medoid for another is significant, with the time complexity of this one sub-operation being $O(n)$, where n is the number of objects present within the given database. When one considers this operation in context - that the entire set of data objects need to be iterated over each time a swap, or potential swap, occurs - the overhead is evident.

A suitable customisation to the K-Medoids procedure is an attempt to limit the number of times that data objects need to be processed, per iteration of swaps of medoids. This suggested concept requires the clustering associations, and thereby the distances from a given data object to its related medoid, be recalculated at most only once when a medoid object is changed. At present, the algorithm recomputes the cluster and medoid assignments for all points in two instances: initially or after a swap has occurred, and also in order to determine the total cost of swapping a medoid within the system. In many situations, this would be grossly inefficient as most points should and will remain associated with the same medoid. The only data instances that would change are those which have close to dual-medoid membership, given the location of the new, potential medoid. With the suggested customisations, the algorithm only recalculates cluster membership after a change has been made, and rather than recomputing all memberships at the second instance, it instead swaps a given medoid out temporarily with the other randomly selected medoid.

Because the second iteration through the entire data set is made redundant, the act of swapping medoids can be performed at the same time that clustering assignments are made. The result of this change is a significant improvement in time complexity, reducing the original system from $O(2n)$ to $O(n)$, whereby n is the number of objects within the given set of data. Put into practice, the results of experiments performed on this specialised algorithm are shown in Section 5.

5. Results and Discussion

5.1. Experimental details

The experimentation detailed in this section was carried out within the *Waikato Environment for Knowledge Analysis* (WEKA) suite for machine learning. This software, developed in the Java programming language, offers a powerful testing harness for analysis of various data mining concepts and implementations. In addition, it offers the ability to extend the suite with additional modules, and in the case of the presented work, clustering methods.

Building upon the functionality that already exists within WEKA, several existing techniques - specifically, DBSCAN and K-Medoids - were implemented, as were the customisations detailed in Section 4. Once developed, these clustering algorithms were applied to the specified test set of forest fire data to determine their effectiveness in terms of the number of clusters found and noise points classified. Efficiency was also monitored in terms of running time com-

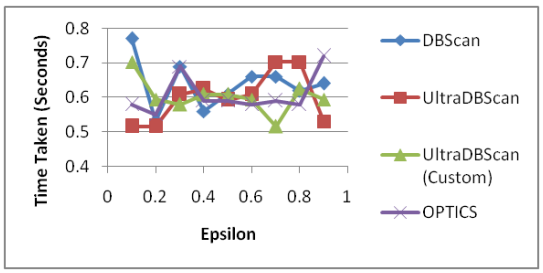

Figure 5: Density-based performance by epsilon

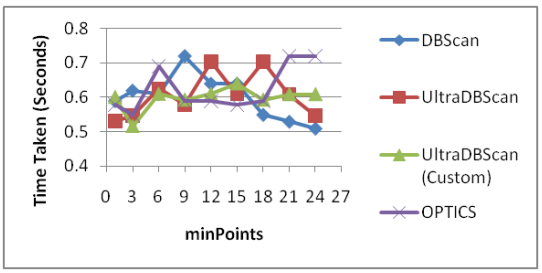

Figure 6: Density-based performance by *minPoints*

plexity. For each different input and clustering technique, the effects of the previously-mentioned customisations were also noted, and corresponding results recorded. Analysis of these results aimed to demonstrate whether the modified algorithms were able to offer any degree of improvement over their original counterparts.

Each test was performed on the same computing hardware in order to ensure objectivity - a recent-model desktop computer with a 2.5GHz Intel Centrino dual-core processor and 3GB of RAM, running the latest stable release of the Java Virtual Machine (JVM 1.6). These experiments examined a range of varying input parameters to determine their effect on an algorithm's results, and to determine any changes in speed.

It should be noted, however, that whilst the results of the experiments described may appear conclusive, there are a number of other factors that may have a significant impact upon the generated output. For example, consider the performance measure of a given clustering technique. In order to perform this analysis, the algorithms need to be run on a computer which, due to operating system overhead, may skew results as a result of other tasks consuming processor load. Similarly, given that the algorithms detailed here are implemented using Java, the JVM or WEKA suite itself may implement some degree of caching, such that the results of one test may unknowingly become the first preprocessing step for the next. These issues can be overcome, however, through the consideration of generalised trends and average results over a number of different tests.

5.2. Comparison results

Density-based Performance The four clustering algorithms used concern density-based techniques: the original DBSCAN algorithm that was implemented within WEKA, the newly implemented DBSCAN algorithm (*UltraDBScan*), its customised counterpart described in Section 4.1 (*UltraDBScan (Custom)*), and finally, the WEKA implementation of the OPTICS method.

Figure 5, 6 and 7 show a detailed comparison of the performance of these density-based clustering algorithms. The first demonstrates the speed of clustering based upon different ε distance values, given the expectation that a larger ε value would result in generally larger clusters being found. The second graph shows that whilst there are minor fluctuations from the change in *minPoints,* the overall trend appears to be that as the value increases, the time taken to achieve results decreases. The exception to this appears to be the OPTICS algorithm, which was otherwise essentially the same throughout. The final figure examines the change in speed for each method given an increasing number of attributes that need to be processed.

These first set of tests demonstrate that for the given data set, the processing time is generally the same for each clustering algorithm, irrespective of a change in ε. Similarly, the second set of testing shows that the processing time for each technique increased at a similar rate as the number of attributes grew. The tests demonstrate that, on average, the original DBSCAN algorithm has the worst performance of each of the algorithms for increases in both ε distance and the number of attributes present. Notably, the new implementation of UltraDBScan was able to typically perform faster than that of the original implementation that ships within WEKA. On closer inspection of the source code behind the original algorithm, it was clear that a difference in how each traverses to a given point's neighbours was the cause of the performance improvement. The two remaining techniques, the customised UltraDBScan and OPTICS, were able to outperform the rest. The reasoning for the customised algorithm performing better than its original counterpart is the addition of the dynamic ε value. As this value essentially constricts the allowable spread of clusters, fewer recursive calls are required to the objects surrounding each cluster. This reduction in processing complexity produces an algorithm that is typically faster, overall.

Partitioning-based Performance The three clustering algorithms used for these tests focus on partitioning-based techniques: the K-Means algorithm, which ships as a standard algorithm within WEKA, the newly implemented K-Medoids algorithm, and its customised counterpart - de-

scribed in Section 4.2, referred to as *K-Medoids (Custom)*.

Figure 8 and 9 show a performance analysis of these partitioning-based clustering algorithms. Figure 8 shows that changing the number of clusters to be defined (k) effects the performance of each method; as expected given that a larger value of k will result in more clustering calculations being performed. In order to test how well each clusterer handles a higher level of dimensionality, Figure 9 shows the test results from changing the number of attributes of each data object processed.

The first set of tests highlight that as the value of k increases and more clusters are calculated, the time needed to process the data set similarly increases. It is evident that the K-Medoids algorithm is the worst-performing of those tested, with the time taken growing extremely quickly for higher values of k. This time complexity can be attributed to the number of calculations that need to take place for each cluster examined. Thus, these results are to be expected from the standard K-Medoids algorithm. However, the modified version of K-Medoids was shown to be the most robust. This demonstrates that the changes suggested have a notable effect on processing time, given that far fewer recalculations are required for a given data set. Finally, the K-Means algorithm ranked between the other two methods, and showed a gradual increase in processing time across testing. Most notably, this rate change was less than that of K-Medoids, but evidently greater than that of the customised algorithm.

Trends in the results when changing the number of attributes of each data object differed somewhat from that of modifying the value of k. In these tests, the two flavours of K-Medoids algorithms performed equally well, with the customised version, as expected, slightly out-performing that of its original version. The K-Means algorithm was slower, given the change in parameters. It should be expected that each of these algorithms suffers a decrease in performance as the dimensionality of the given data objects becomes higher, and this is evident in the results shown. The increase in the time taken for the K-Means algorithm to complete most probably relates to a difference in implementation, rather than a fundamental change in the way each algorithm handles attributes and distances.

Clustering Results The ability for each of the density-based algorithms to detect noise was also examined. The results for each of the density algorithms are described within Figure 10 and 11.

The overall trend described by these tests demonstrates the expected behaviour of any density-based technique: as the distance checked for neighbours increases, the number of points classified as noise decreases. In addition, this directly affects the size of the clusters discovered. Similarly, as the value of *minPoints* increases, the number of points

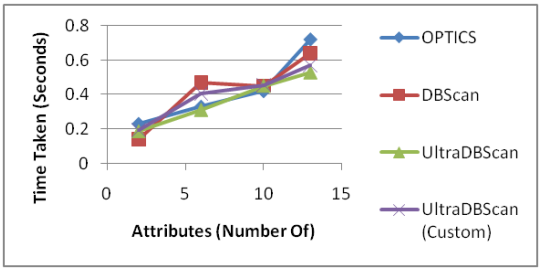

Figure 7: Density-based performance by attributes

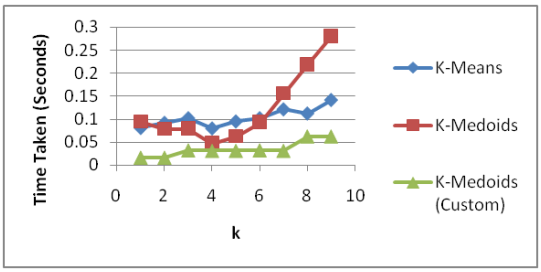

Figure 8: Partitioning performance by k

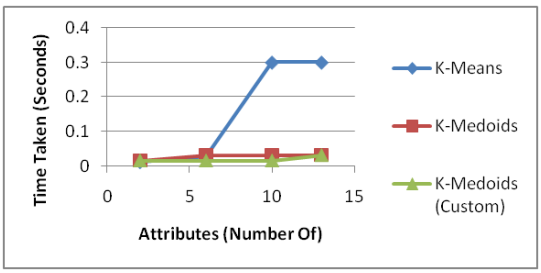

Figure 9: Partitioning performance by attributes

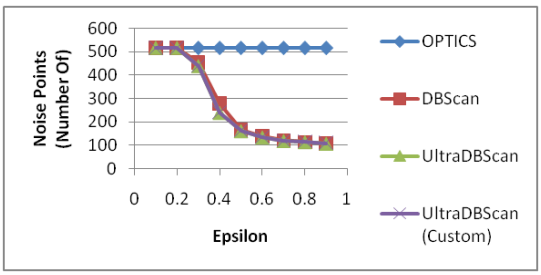

Figure 10: Density-based performance by resulting noise from change in epsilon

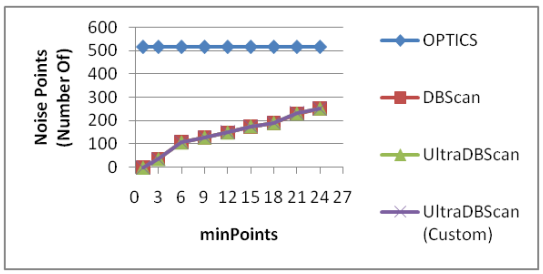

Figure 11: Density-based performance by resulting noise from change in *minPoints*

being classified as noise increases also.

The results demonstrate that all techniques - with the exception of OPTICS which considered all points to be noise for all values of ε - were able to detect similar amounts of noise, and thus define a similar number of clusters. The exception to this trend was the customised DBSCAN algorithm which, for most ε distances, and values of *minPoints*, tested, discovered a slightly greater amount of noise, or a greater number of clusters. The reasoning for these changes is directly associated with the dynamic change of ε, as the size of clusters grew throughout the processing of the database. Having this modification present also demonstrated that the number of points assigned to certain clusters changed also. For instance, with a *minPoints* value of 1, the size of several clusters changed considerably, thus illustrating the the dynamic ε value is having an effect. In comparison, the same cluster was defined as only a single point for the other methods. Thus, these results demonstrate that, depending on the application, this type of interchangeable variable may bring about more-suitable clustering assignments.

Overall, the original WEKA implementation of DBSCAN performed comparably with the newly implemented UltraDBScan for all tests. This result demonstrates that not only was the new implementation able to compute clusters correctly, as it was able to find the same amounts of noise and clusters, but that the provided algorithm was not as efficiently written.

Due to comparable limitations of the K-Medoids and K-Means algorithms, a similar figure cannot be produced. The inherent purpose of these partition-based techniques is to ensure that all elements in a database are assigned to a cluster, and thus none are discarded as noise. This is described in the operations in Algorithm 2 and 3, whereby each object present is assigned to its nearest medoid or centroid. Unlike the density-based techniques described here, no degree of analysis occurs to determine if a given point should legitimately belong to any cluster.

5.3. Issues

As is to be expected of any data mining procedure, the best results from a clustering operation depend entirely upon many different factors. These include, but are not limited to, the actual data set being processed, the clustering technique selected, and the input parameters provided to the technique. In some situations, a certain algorithm may determine suitable clusters within one data set, but may be completely ineffective with another. Likewise, using unsuitable input parameters for a given clustering process may result in similarly unsuitable cluster assignments for data instances. As a result, all of the aforementioned aspects of data mining need to be considered before an algorithm is chosen, and the application of the technique should be performed by a person who has sufficient domain knowledge to make correct decisions that will reveal suitable output.

Furthermore, as detailed in Section 2, the most suitable types of data are detailed for each given subset of clustering techniques. It should be noted that the results demonstrated in this paper summarise tests executed within the given testing environment, and only represent output associated with the single data set of forest fire information. described in Section 1.4. As mentioned previously, using a different set of data may, and most likely will, reveal different patterns and see a change in processing time complexity. This would be especially evident if incorrect input parameters are used for the given clustering algorithms. However, the results detailed in Section 5.2 indicate that for the data set under consideration, the DBSCAN and K-Medoids algorithms (and their customised counterparts) reveal useful clusters. From this, it can be generally inferred that for similar types of data, these same algorithms should result in the definition of similar clusters in a similar processing time-frame.

6. Conclusion

The results demonstrate that for density-based algorithms, an incorrect or non-optimised selection of ε distance can significantly alter the clusters defined. However, a change in ε distance had very little effect on the efficiency of the technique. This outcome is caused by the way in which this type of clusterer progresses through the database. Whilst some algorithms may utilise recursion to examine points, the time complexity essentially resolves as $O(n \log n)$, where n is the number of objects in the database. This occurs because each point only needs to be checked once, and only compared to its own neighbours. Thus, any effect caused by either of the input variables, *minPoints* or ε, will be negligible. This stands in contrast to the change in performance seen when the *k* value is altered for partitioning-based clustering techniques. As this value directly affects the number of calculations that need

be performed on each data object, searching for increasing numbers of clusters has an inflationary effect on the time required to process a data set.

A notable conclusion that can be drawn from the results is that an increase in the number of attributes being processed will see a decrease in the processing speed for all given clustering types tested. The cause of reduction in performance is related to the time taken to compute distances between each of the objects that need to be compared. In lower dimensions, when few attributes are present, calculations are simple as the results have shown. However, as higher dimensions need to be considered, the time complexity increases accordingly.

The given data set demonstrates potential for both of the customised algorithms. For UltraDBScan, the set showed that different quantities of noise could be found. Whilst not strictly applicable to this specific data set, this ability to locate additional noise could greatly affect the clustering results for data from other domains. As an example, consider several clusters with a *bridge* of objects between them. As the algorithm begins to cross the bridge from an initial cluster, it will decrease the distance it checks for neighbours accordingly, essentially severing the link, and declaring each point of the bridge as noise.

Whilst not quantifiable, due to the difficulty of attempting to compare one cluster to another, the improvement suggested in this paper to K-Medoids is similarly useful. As this customised method no longer needs to re-analyse the entire data set twice for each iteration, the time complexity decreases significantly, and results in a technique that is far more robust as an increasing number of clusters is found.

Overall, the different methods of clustering have been shown to produce varying levels of performance and yield different degrees of usefulness from results. Whilst there are similarities between the clustering algorithms and their generalised types, the overall results produced show that a choice of technique is one of the most important decisions to be made in clustering.

7. Future Work

As data mining, and specifically clustering, becomes more of a part of every-day business and organisational operation, the necessity for faster and equally accurate algorithms rises. The improvements suggested in this paper, while novel and logically presented, are not entirely comprehensive and thus scope for further ongoing research exists in several areas. There are many other potential improvements that could be investigated within the field of clustering techniques - extensions to the work presented here are essentially unbounded.

Partitioning techniques (including the suggested UltraK-Medoids algorithm) require that all points in the data set are allocated to a cluster, unlike density-based methods which can leave noise points in an unclustered state. This creates varying levels of noise sensitivity across the K-algorithms, and research into development of a method that works on both principles would be beneficial.

The requirement that both partitioning and density-based techniques specify input parameters places certain limitations on the effectiveness of the algorithms when these values are not optimised. Choosing the values correctly takes both experimentation and time, which adds overhead to the process of drawing useful conclusions from mined data. Partitioning methods require that the number of clusters to be found is statically set, and incorrect choice of this value can lead to serious noise sensitivity. Investigation of the advantages of allowing the algorithm to dynamically choose the number of clusters (as used in density-based techniques) is thus justified. Density-based methods suffer from incorrect choice of ε neighbourhood radius and minimum points per cluster. UltraDBScan has addressed the issue of ε selection by allowing the algorithm to dynamically chose this value as it progresses through points in the data set under analysis. However, more research into managing the effect of the minimum points variable would be clearly beneficial. By reducing or eliminating the effects these variables have on results, a system will be able to produce more objective output with greater accuracy.

Given that the algorithms this paper discusses have been in existence for up to 20 years, and that there have been various papers and theses produced regarding the aforementioned issues, the suggestions made in this section appear as the logical path for future investigation. Others can extend this work by seeking alternative ways of improving the semantics of the proposed algorithms, rather than attempting to improve the actual operations which have been already been fine tuned through the extensive research summaries in section 2.

A. Implementation

This paper presents a method for developing for classes for WEKA, using a Java integrated development environment (IDE) named IntelliJ. Developed by a company named JetBrains, IntelliJ has been found to be one of the most powerful and useful development environments, and as such, these instructions are specific to this program. They may be extended, with some investigation, to other development environments, such as NetBeans. However, no assurances can be placed upon whether the same procedure here can be replicated using another program.

Follow these guidelines to easily create and further develop clustering algorithms, classifiers and association rules processes for use within WEKA:

1. Install the latest version of IntelliJ from

`http://www.jetbrains.com/idea/`. It is a commercially licensed piece of software, so a developer either needs to use the trial version, or otherwise purchase or obtain a license key. You will also need a Java Development Kit (JDK); version 1.6 is the latest as of the time of writing.

2. Install the latest version of WEKA from `http://www.cs.waikato.ac.nz/ml/weka/`.
3. Create a new IntelliJ project, and set up your JDK accordingly.
4. Set up your project so that you can use the WEKA classes within your own.
   (a) Go to File ⇒ Settings ⇒ Project Settings ⇒ Global Libraries
   (b) Add a new *Global Library* and call it something like "WEKA"
   (c) Attach WEKA's JAR file by selecting "Attach Classes" and locating the relevant file. You will find this located in your WEKA install location. You may wish to make a copy or move this JAR into your own project folder.
   (d) Click onto "Apply" to save the changes. The library will be parsed accordingly and you can now access and utilise WEKA libraries in your own code.
5. Add a run configuration for WEKA, so you can run and debug your code from IntelliJ.
   (a) Go to Run ⇒ Edit Configurations
   (b) Click onto the "Add" button, and select "Application"
   (c) Under "Main class" enter `weka.gui.Main`
   (d) Under "VM Parameters" you can control the JVM environment. Using a value such as `-Xmx512m` will allocate the JVM 512MB of RAM, rather than 128MB or less that would otherwise be the default. Tweaking these settings is especially useful if you find your data set is too complex and it causes the JVM to run out of memory.
   (e) You may wish to change your "Working directory" to be where your data sets reside, for ease of access. This location specifies where the default folder is for when you open any file dialogs within WEKA.
   (f) Ensure that your *classpath* is specified to be that of your current project so your classes can be found within WEKA.
   (g) Ensure that under "Before launch" the option "Make" is *not* ticked. If it is, your classes will not be visible as your build location will be inadvertently cleaned when you run WEKA.
   (h) Click "OK" to save the changes.
6. Make your classes visible within WEKA requires an extra step, as classes are found dynamically in the latest version of the data mining suite.
   (a) Locate the previously-used JAR file for the WEKA library - it is most likely in your WEKA install location.
   (b) Open it within your favourite archive extraction utility, noting that you may need to rename the `.jar` to `.zip`, if your program does not support the `.jar` format.
   (c) Enter the "weka" folder, and then the "gui" folder.
   (d) Find the `GenericPropertiesCreator.props` file and extract this into the base of your user home directory.
   (e) Edit the file in a text editor, and near the top, locate the line `UseDynamic=false`.
   (f) Change the `false` to `true` and your classes, once created correctly, will be accessible from within WEKA.
7. Build your classes, taking the following into account:
   (a) Make each of your classes part of the `weka` package. For example, if you were creating a clusterer, like the ones described in this paper, use the `weka.clusterers` package.
   (b) Make each of your classes extend a certain interface or abstract class from WEKA. This is necessary to allow WEKA to dynamically find your classes, and IntelliJ will automatically fill out which methods need to be implemented and extended. For example, you can use DensityBasedClusterer for this type of clusterer, or RandomisableClusterer for those that need some degree of randomisation in their processing.
8. When you want to test your classes, go to Build ⇒ Rebuild Project. Your classes will be compiled in your build directory. Then, run the WEKA configuration from IntelliJ. WEKA will start, and your new classes can be tested.

This introduction has introduced the method of creating new classes for the WEKA data mining environment. Further documentation is available, as at the time of writing,

from `http://weka.wiki.sourceforge.net/`. The various resources at this website explain details at even more depth on how each part of your code is used in WEKA.

References

[1] C. Aggarwal, J. Wolf, P. Yu, C. Procopiuc, and J. Park. Fast algorithms for projected clustering. *ACM SIGMOD Record*, 28(2):61–72, 1999.

[2] M. Ankerst, M. Breunig, H. Kriegel, and J. Sander. OPTICS: ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 49–60. ACM New York, NY, USA, 1999.

[3] P. Berkhin. Survey of clustering data mining techniques.

[4] M. Berry and M. Browne. *Lecture Notes in Data Mining*. World Scientific, 2006.

[5] P. Cortez and A. Morais. A Data Mining Approach to Predict Forest Fires using Meteorological Data. *New Trends in Artificial Intelligence, Proceedings of the 13th EPIA 2007 - Portuguese Conference on Artificial Intelligence*, pages pp. 512–523, 2007.

[6] M. Ester, H. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press*, pages 226–231, 1996.

[7] U. Fayyad, R. Uthurusamy, et al. Data mining and knowledge discovery in databases. *Communications of the ACM*, 39(11):24–26, 1996.

[8] J. Grabmeier and A. Rudolph. Techniques of Cluster Algorithms in Data Mining. *Data Mining and Knowledge Discovery*, 6(4):303–360, 2002.

[9] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, second edition, 2006.

[10] J. Hartigan and M. Wong. A K-means clustering algorithm. *JR Stat. Soc. Ser. C-Appl. Stat*, 28:100–108, 1979.

[11] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means Clustering Algorithm. *Applied Statistics*, 28(1):100–108, 1979.

[12] A. Hinneburg and D. Keim. An efficient approach to clustering in large multimedia databases with noise. *Knowledge Discovery and Data Mining*, 5865, 1998.

[13] A. Jain, M. Murty, and P. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3), 1999.

[14] G. John. Behind-the-scenes data mining: a report on the KDD-98 panel. *ACM SIGKDD Explorations Newsletter*, 1(1):6–8, 1999.

[15] G. Karypis, E. Han, and V. Kumar. Chameleon: Hierarchical Clustering Using Dynamic Modeling. *COMPUTER*, pages 68–75, 1999.

[16] Kaufman and Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, 1990.

[17] L. Kaufman and P. Rousseeuw. Clustering by means of medoids, 405–416. *Statistical Data Analysis Based on the L1-Norm and Related Methods*, 1, 1987.

[18] L. Kaufman and P. Rousseeuw. Partitioning Around Medoids (Program PAM). *Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, NY*, pages 68–127, 1990.

[19] E. Kolatch. Clustering algorithms for spatial databases: A survey, 2001.

[20] D. Moore. Probability and Statistics in the Core Curriculum. *MAA NOTES*, pages 93–98, 1998.

[21] R. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proceedings of the International Conference On Very Large Databases*, pages 144–144. Institute of Electrical & Electronics Engineers (IEEE), 1994.

[22] J. Sander, M. Ester, H. Kriegel, and X. Xu. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

[23] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In *Proceedings of the International Conference On Very Large Databases*, pages 428–439. Institute of Electrical & Electronics Engineers (IEEE), 1998.

[24] W. Wang, J. Yang, and R. Muntz. STING: A Statistical Information Grid Approach to Spatial Data Mining. In *Proceedings of the International Conference On Very Large Databases*, pages 186–195. Institute of Electrical & Electronics Engineers (IEEE), 1997.

[25] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: A New Data Clustering Algorithm and Its Applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.