A Parameterless Method for Efficiently Discovering Clusters of Arbitrary Shape in Large Datasets

Andrew Foss
University of Alberta, Canada
afoss@cs.ualberta.ca

Osmar R. Zaïane
University of Alberta, Canada
zaiane@cs.ualberta.ca

Abstract

Clustering is the problem of grouping data based on similarity and consists of maximizing the intra-group similarity while minimizing the inter-group similarity. The problem of clustering data sets is also known as unsupervised classification, since no class labels are given. However, all existing clustering algorithms require some parameters to steer the clustering process, such as the famous k for the number of expected clusters, which constitutes a supervision of a sort. We present in this paper a new, efficient, fast and scalable clustering algorithm that clusters over a range of resolutions and finds a potential optimum clustering without requiring any parameter input. Our experiments show that our algorithm outperforms most existing clustering algorithms in quality and speed for large data sets.

1. Introduction

There exist a multitude of algorithms for clustering data. Basically, they each try to concentrate on some important issues in clustering, such as high dimensionality problems, efficiency, scalability with data size, sensitivity to noise in data, identification of clusters with various cluster shapes, etc. However, none has managed to take all these factors into account at once. The major drawbacks of existing clustering algorithms include the splitting of large genuine clusters, which is the case for partitional approaches such as k-means [6]; failure to handle convex and elongated shapes of clusters as is the case with most hierarchical approaches such as CURE [11] and ROCK [4]; and the sensitivity to noise in the data such as in CHAMELEON's case [7] or DBSCAN [2], a density-based clustering algorithm.

The greatest difficulty in the field of data clustering is the need for input parameters. Many algorithms, especially the hierarchical methods [5, 11], require the initial choice of the number of clusters to find. Even where this is not required or the algorithm can stop automatically before that number is reached, other parameters greatly influence the output. One can argue that for some applications parameters are a means to incorporate domain knowledge into the clustering process and thus are beneficial in some circumstances. This is particularly true when the number of clusters to be discovered is predetermined and fixed by the application. However, in many applications the optimal values of these parameters are very difficult to determine. Often a long and tedious trial-and-error process is used to tune these parameters. However, when the number of dimensions is larger than three, this becomes extremely difficult or unpractical, and an automated process is desirable. On the other hand, domain knowledge could be expressed in the definition of similarity functions used to measure how similar or dissimilar data points are in order to group them in clusters.

In this paper, we present a novel clustering algorithm called *TURN** which does not require the input of any parameter and still efficiently discovers clusters of complex shapes in very large data sets. We report the efficiency and demonstrate the effectiveness of *TURN** on large and complex data sets containing points in 2D space borrowed from [7, 13] for comparison reasons.

Section 2 gives an overview of existing, well-known clustering algorithms and Section 3 describes our clustering algorithm *TURN**. Section 4 presents some experimental results for *TURN** and six established clustering algorithms. Finally, Section 5 concludes and discusses future work.

2. Related Work

There are mainly four groups of clustering methods: partitioning methods, hierarchical methods, density-based methods and grid-based methods. We give a brief introduction to these existing methods in this section.

Supposing there are n objects in the original data set, partitioning methods break the original data set into k partitions. The basic idea of partitioning is very intuitive, and the process of partitioning is typically to achieve certain optimal criterion iteratively.

The most classical and popular partitioning methods are k-means and k-medoid, where each cluster is represented by the gravity centre of the cluster in k-means method or by one of the "central" objects of the cluster in k-medoid method.

All the partitioning methods have a similar clustering quality and the major difficulties with these methods include: (1) The number k of clusters to be found needs to be known prior to clustering requiring at least some domain knowledge which is often not available; (2) it is difficult to identify clusters with large variations in sizes (large genuine clusters tend to be split); (3) the method is only suitable for concave clusters.

A hierarchical clustering algorithm produces a dendrogram representing the nested grouping relationship among objects. In the past few years, many new hierarchical algorithms have been published. The major difference between all these hierarchical algorithms is the measure of similarity between each pair of clusters and the underlying modelling of the clusters. Because these algorithms are typically computationally expensive, many proceed by sampling the data and clustering only a representative sample of the data points, which puts the effectiveness of the clustering at the mercy of the goodness of the sampling method.

Instead of using a single point to represent a cluster as in centroid/medoid based methods, CURE [11] uses a constant number of representative points to represent a cluster. These are selected so that they are well scattered and then "shrunk" towards the centroid of the cluster according to a shrinking factor α. Cluster similarity is measured by the similarity of the closest pair of the cluster's representative points The problem with CURE is its global similarity measure which makes it ineffective with complex data distributions and it can cause false outliers (See Figure 4).

The same authors developed ROCK [4]. ROCK operates on a derived similarity graph and is thus suiteable for both numerical and categorical data. ROCK measures the similarity between pairs of clusters by the normalized number of total links between them. This is given as a fixed global parameter (θ). However, ROCK is extremely sensitive to θ, which reflects a fixed modeling of clusters. ROCK's clustering result is not good for complex clusters with various data densities, and the algorithm is sensitive to noise.

A recent state-of-art clustering method is CHAMELEON [7] which considers both relative connection and relative closeness. The algorithm operates on a derived similarity graph allowing the clustering of numerical as well as categorical data like ROCK. Phase one of the algorithm uses a graph partitioning method to pre-cluster objects into a set of small clusters. The second phase merges these small clusters based on their similarity measure. CHAMELEON has been found to be very effective in clustering convex shapes. However, the algorithm cannot handle outliers and needs parameter setting in order to work effectively.

Density-based methods identify clusters through the data point density and can usually discover clusters with arbitrary shapes without a pre-set number of clusters. DBSCAN is a typical density-based method which connects regions with sufficiently high density into clusters. Each cluster is a maximum set of density-connected points. Points are connected when they are density-reachable from one neighbourhood to the other. A neighbourhood is a circle of a radius (ϵ) and reachability is defined based on a minimum number of points ($MinPts$) contained in a radius ϵ. DBSCAN, however, is very sensitive to the selection of ϵ and $MinPts$ and it cannot identify clusters with different densities. The problem of discovering clusters with different densities and clusters within clusters was alleviated in OPTICS [9] by the same authors. However, the parameter $MinPts$ still needs to be defined.

Grid-based methods first discretize the clustering space into a finite number of cells, and perform clustering on the gridded cells. The main advantage of grid-based methods is that the processing speed only depends on the resolution of griding and not on the size of the data set. The grid-based methods are more suitable for high density data sets with huge number of data objects in limited space.

Representative grid-based algorithms include CLIQUE [1] and, recently, WaveCluster [12] which applies a noise filter and wavelet transformation. Grid-based quantization of the data space speeds up the processing, but due to the rectangular structure, the algorithm represents a much coarser approach to levels of resolution than that adopted by our algorithm *TURN** presented herein, or DBSCAN, and is much more likely to lead to the misclassification of data points. The algorithm requires this quantization so one does not have the option of classifying the data at full resolution. WaveCluster offers multiresolution by skipping "rows" of the quantized data (i.e. down sampling), a much cruder approach than that used by the other algorithms that use all the data and simply expand the nearest neighbour definition or equivalent of each point. WaveCluster's main benefits are speed and scalability since the data is read in and quantized and then subsequent processing is on a much smaller effective data size.

While WaveCluster claims to be parameter free, our research showed that its clustering results are quite sensitive to the settings that have to be made. In fact [12] points out that knowing the number of clusters to be found is very helpful for choosing the parameters for WaveCluster.

2.1. Unsupervised Method: TURN

Previously, we have devised a non-parametric approach to categorical data clustering in a non-Euclidean space called TURN that we used for clustering web access transactions [3]. Here we present a non-parametric approach suited to spatial data as well as higher dimensional spaces. TURN operates on a derived similarity graph allowing any similarity function to be plugged in. The basic idea of TURN is to identify natural boundaries between clusters instead of spotting clusters themselves. This amounts to searching for the minima in the distribution of points, which represent turning points. TURN proceeds by iteratively selecting a non assigned object p and computing the similarity between p and all non-assigned objects. After sorting the similarities and differencing them 3 times, the algorithm looks for a change of sign (i.e. turning point) in the differenced number series. All objects that appear before the turning point are assigned to the same cluster as p and their direct neighbours are also pulled in. The process repeats until all objects are assigned [3]. TURN has been tested on categorical data only and experimentally it outperformed ROCK in clustering web usage data.

3. TURN* Algorithm

In this section we introduce our new clustering algorithm *TURN**, a non-parametric clustering approach that efficiently discovers clusters of arbitrary shapes.

*TURN** consists of an overall algorithm and two component algorithms, one, an efficient resolution dependent clustering algorithm TURN-RES which returns both a clustering result and certain global statistics (cluster features) from that result and two, TurnCut, an automatic method for finding the important or "optimum" resolutions from a set of resolution results from TURN-RES. To date, clustering algorithms have returned clustering results for a given set of parameters, as TURN-RES does, and some have presented graphs or dendograms of cluster features from which the user may be able to adjust or select the parameters to optimize the clustering. *TURN** takes clustering into new territory by automating this process removing the need for input parameters. Clustering Validation is a field where attempts have been made to find rules for quantifying the quality of a clustering result. Though developed independently, TurnCut could be seen as an advance in this field which has been integrated into the clustering process.

3.1. TURN-RES: Clustering at one resolution

This is a fast, efficient, scaleable clustering algorithm for a single resolution. While little discussed, except in papers such as [12], resolution is a key concept in clustering. When a radius is defined, as in DBSCAN, or some related parameter, a particular view is being set that has an equivalence to viewing a density plot with a microscope or telescope at a certain magnification. The night sky is one example; as the magnification level is adjusted, one will identify different groupings or clusters. The CHAMELEON data set (Figure 1) is another example. It looks like there are nine clusters but given a magnifying glass, the large clusters will be seen to have their own sub-clusters.

TURN-RES is resolution or scale dependent because of its definition of "close" neighbours. Two points are considered "close" only if they are separated by a distance $d \leq 1.0$, at a given resolution, along all dimensional axes. For example, points x_0 (1, 1) and x_1 (3, 3) are not neighbours unless the scale is reduced (e.g. $\div$ 2 giving x_0 (.5, .5) and x_1 (1.5, 1.5)). We compute a density value t_P for each point P based on its distance to its nearest neighbours, irrespective of their closeness, which, in our approach, is a maximum of 2 per dimension, one on each side. t_P allows us to determine how closely packed the points are locally and a threshold ϕ is set as a cut-off to differentiate between points that are to be treated as internal or external to a cluster. Points that fall on the edge of a cluster will not be marked as internal but they get included because close neighbours to internal points are pulled into the cluster. ϕ is not a function of the data set but rather of the method employed and a single value sufficed for all data sets tested so this is not a user input parameter.

As noise typically can create small clusters, we defined small clusters as noise or outliers. Small is defined as min$(100,\ N/100)$ where N is the number of data points. TURN-RES collects certain global statistics or cluster features, being k, the number of clusters, n, the number of points assigned to clusters (not considering outliers), t, total density and t_m, mean density ($t \div n$). Our interest is to characterize the resolution level in such a way that levels of particular interest can be determined automatically. t is the sum over n of t_P that is defined for point P as:

$$t_P = 1 \div \sum_{i=1}^{2} \sqrt{(d_{L_i}^2 + d_{R_i}^2)} \qquad (1)$$

Note: d_L and d_R are the distances to the 'left' and 'right' side neighbours along a dimensional axis i.

Once the data points have been characterized as internal or external, they are simply agglutinated into clusters. An unclassified internal point is selected, a new cluster number is assigned to it and all of its "close" neighbours are similarly classified. For each of those that are also internal, the process is repeated. Not incorporating "close" neighbours of external points stops clusters growing across 'noise' bridges.

To quickly determine the nearest neighbour ids of each point a single sort is performed on each dimension. Building the clusters requires one further sort by id. Each sort is followed by a single scan giving a computational complexity of $O(Nlog(N))$.

TURN-RES Algorithm

Input: 2D data points and resolution level r

Output: Clustered data points

```
1. For each data point $P$, scale coordinates of $P$ to resolution $r$
   and find the two nearest neighbours on each dimensional axis,
   and the distance $d$ of each from $P$; if $d \leq 1.0$ assign the point
   as a "close" neighbour;
2. compute the density $t_P$; set $P$ as internal if $t_P > \phi$ ($\phi$ is fixed
   and not an input parameter)
3. For each non assigned internal data point $P$ do
   (a) add $P$ to new cluster $C$;
   (b) add all "close" neighbours of $P$ to $C$;
   (c) for all $m$ added points that are internal add all their
       "close" neighbours to $C$ and repeat until $m = 0$;
```

TURN* **Algorithm**

Input: 2D data points

Output: "Best" clustering of data points

```
1. start at resolution $r$=1:1. Seek $S_\infty$ by increasing/decreasing the
   resolution by step $p_{2.5}$ (multiplying or dividing by $2.5, 5.0, ...$)
   and clustering at each resolution until all points are labelled as
   outliers;
2. scan from $S_\infty$ towards $S_1$ ($k = 1$) by repeating.
   (a) decrease the resolution $r$ by step $p_{0.5}$;
   (b) cluster at $r$ with TURN-RES;
   (c) store clustering result and statistics $t$, $t_m$, $k$, and $n$ for $r$;
   (d) stop if $k = 1$ else call TurnCut to determine if $r$ is an
       "optimum" clustering result and stop on success.
```

3.2. TurnCut

TurnCut uses the core of the TURN algorithm described in Section 2.1, from our previous work in [3], detecting a change in the third differential of a series to identify important areas in a cluster feature across resolution series built by repeated calls to TURN-RES by the *TURN** algorithm. Single and double differencing are routinely used in time series analysis ([10]) to render a series stationary. Differencing amounts to a highpass filter which we have employed, differencing thrice (double differencing the change values of the series), as we found it to be most effective way to reveal meaningful change in the underlying trend.

Though developed independently of work on Clustering Validation, TurnCut automates other authors' concept of finding the "knee" in the cluster feature graph [8]. It picks out the first (and subsequent) "abrupt" change in the overall trend of the curve - acceleration or reversal of the rate of change of the clustering feature studied. If the data points being clustered are homogenously distributed, no "turn" will be found. If clusters exist, TurnCut will pick out the point where stabilization occurs in the clustering process, which will often coincide with a level that an observer would identify as a clustering result (almost by definition - we would never pick out a level that did not appear to represent a certain plateau). In general there can be several of these and the algorithm can find them all even though in this paper the algorithm given for *TURN** stops at the first found. In effect, TurnCut is detecting plateaus in the entropy curve.

Other authors only analyzed a feature versus cluster number (k) graph [8]. We evaluated k and found better results with mean density and often further improvements with total density (as defined above) across resolutions. In most algorithms data can only be collected for a particular value of k. In *TURN** we can study as fine a resolution step size as we choose, several steps often yielding the same value of k. Our result makes sense because each statistic has more information than the previous - k is a fairly coarse statistic compared with those we defined that change with every point added to the clustering result.

3.3. TURN*: Finding the best clustering

The algorithm proceeds by detecting clusters across a range of different resolutions stopping (or at least flagging) what is considered as the "optimal" clustering using TurnCut. A resolution is simply a scale by which all data point values are multiplied (see TURN-RES above).

Naturally, TURN-RES will return a clustering result only within a certain range of resolutions. On one end of the range every data point will be classified as noise/outlier. Moving in the direction of increasing k, the first resolution level at which this occurs we call S_∞. Moving in the other direction a point is reached where all points are included in a single cluster (S_1). First, the algorithm seeks S_∞ starting by clustering with TURN-RES at resolution 1:1 and then stepping out at a large geometric increment ($p_{2.5}$ = $\times 2.5$ where the scale $>$ 1:1; $\div 2.5$ where the scale $<$ 1:1) clustering until S_∞ is found. Then the step size is reduced ($\times .5, \div .5$) and a scan over $S_\infty \rightarrow S_1$ is performed. Geometric steps sizes are used as: a) this ensures quickly finding S_∞ and traversing the range to S_1, and b) optical magnification steps are always given in geometric values.

At each step, TURN-RES is called and the clustering result and global features/statistics are stored and TurnCut is called to assess if an "optimum" clustering has been achieved. If so, either *TURN** stops as we did in this research or it can be allowed to continue, collecting information for all the key resolutions flagged by TurnCut.

No sampling takes place, however the scalability of

*TURN** is evidenced by the performance on small and large data sets (Table 2). The space complexity is straightforward. For each object, a simple data structure is needed to store coordinates in the $D = 2$ dimensional space, the nearest neighbours on each dimensional axis, and some specific data such as cluster label, type of point etc. Thus, the memory space needed is $O(DN)$.

3.4. Parameter Free?

The parameters involved in the component algorithms are 1) ϕ, that defines if a point is to be treated as internal, 2) the resolution level given to TURN-RES, 3) the definition of a "small" cluster, and 4) the step size(s) used between resolutions at which TURN-RES is run.

Our implementation is parameter free in so far as 1) ϕ is part of the concept of closeness for a resolution level and thus should not need to be varied as proved to be the case; 2) *TURN** feeds TURN-RES a series of resolutions starting from the extreme case (S_∞) where all points are identified as outliers so the user is never asked to enter a resolution; 3) we found that only very small data sets would need this modifying and *TURN** is not intended for such sets; 4) the step size starts large until S_∞ is found and is then reduced. In choosing the step size there is a trade-off between fineness and speed. We found the step size we chose to be robust across varied data set types but if the differentiation between cluster densities in the data was small, a key resolution could be missed. However, to be secure against this the algorithm could be extended as follows: Once TurnCut flags a resolution of interest the range across the previous step can be scanned by further reducing the step size giving finer resolving power. It is most unlikely that any user input would be needed in this case and the speed of the algorithm makes this additional processing reasonable.

While there will never be a perfect parameter free solution, our implementation proved robust across a wide range of different data set types with differing cluster densities, closeness of clusters, arbitrary shaped clusters and noise levels including many examples of noise "bridges".

4. Experimental Results

In this section, we present experimental results to evaluate *TURN** and compare the performance of our algorithm with other well-known clustering algorithms: k-Means, DBSCAN, CURE, ROCK, CHAMELEON and WaveCluster. The implementation of DBSCAN was obtained from the authors of the algorithm (University of Munich, Germany), while the implementation of ROCK and CURE was obtained from the authors of CHAMELEON (University of Minnesota, USA) written for the evaluation of their own algorithm [7]. However, as they could not provide the CHAMELEON code for legal reasons, this was implemented locally. We believe that our implementation and optimization of CHAMELEON is similar to the one published by the authors in [7] since we get the same performance on the same data sets as that presented in [7]. WaveCluster [13] was also implemented locally for the same reasons with equal success.

We tested *TURN** on data sets provided by the developers of CHAMELEON [7] and WaveCluster [13]. Due to CHAMELEON's computational intensity, their files are rather small: 8K to 10K. The main benefit claimed in the WaveCluster paper is its effectiveness on large data sets and their data is 100K+ points. We chose these data sets because they are publicly available and they have been used to evaluate other algorithms. Moreover, these data sets are 2-dimensional making it easier to visualize and provide comparisons. We have experimented with various data sets with sizes varying from 8k to more than 575K data points and our algorithm performed well in all cases.

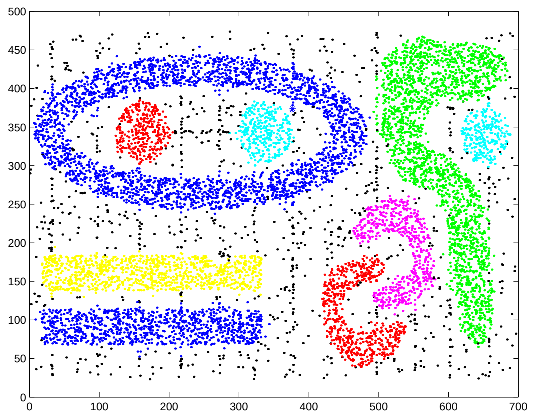

Figure 1. *TURN's clustering result on t7.10k.dat before cleaning**

On a 10K data set (t7.10k) Figure 1, TURN-RES computed a single resolution in 0.26 seconds and the total process of *TURN** to find the optimum resolution took 3.90 seconds. A single run of CHAMELEON took 28 minutes with the parameter $MinSize$ set to 4%. $MinSize$ is the size of the graph partition in the first phase of the algorithm. Selecting a different value would slow down CHAMELEON or degrade the results. The process of finding the correct parameters to give a good clustering result took several hours. DBSCAN is nearly as fast as *TURN** for a single resolution/parameter setting but also required many runs to find the optimal input variables as it is very sensitive to its parameters. We effectively spent hours tuning the parameters and found out that the best values for the t7.10k.dat dataset were $MinSize = 4$ and $\epsilon = 5.9$. This would have been impossible with a higher dimensionality

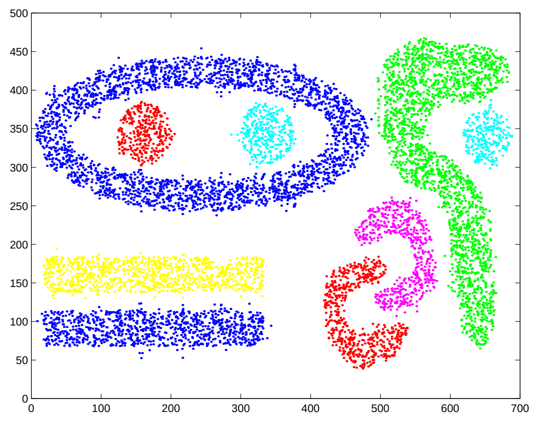
Figure 2. *TURN**'s cleaned clustering result on t7.10k.dat after removal of points identified as noise

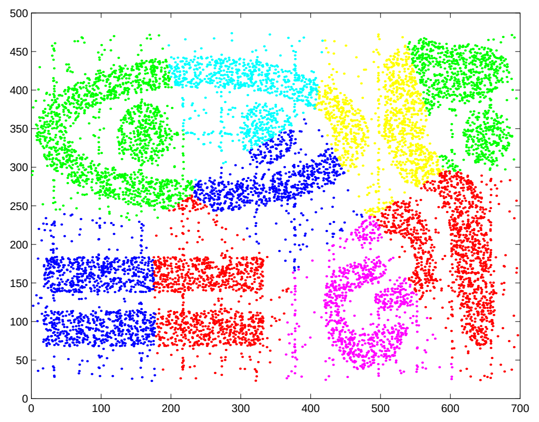
Figure 3. K-means's clustering result on t7.10k.dat with $k = 9$

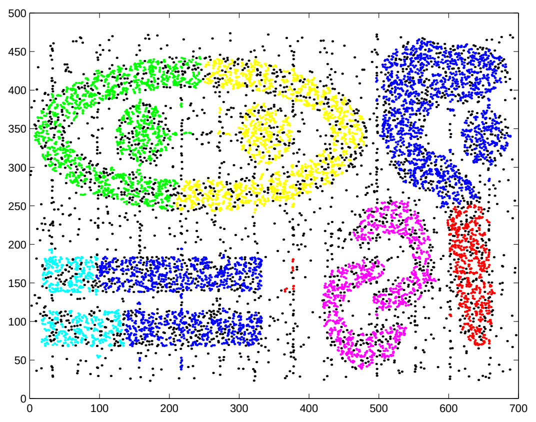
Figure 4. CURE's clustering result on t7.10k.dat with $k = 9$, $\alpha = 0.3$, and number_of_representative_points $= 10$

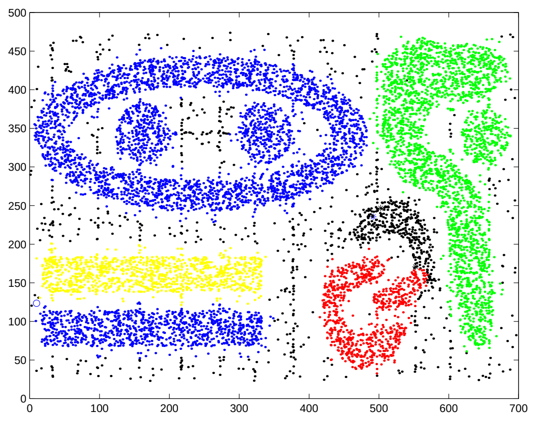
Figure 5. ROCK's clustering result on t7.10k.dat with $\theta = 0.975$ and $k = 1000$

data space since validation is difficult if not impossible.

For the data sets chosen, *TURN** took typically 10-20 resolution tests, performed automatically, to find an "optimum" resolution. In our research, *TURN** found the resolution that a human observer would tend to choose in 80% of cases, and in the other cases, it stopped one or two resolutions away on what was identifiably a meaningful clustering result. It can also be allowed to run on, collecting data at each "turn" or key resolution level building the equivalent of a dendogram that reveals the clustering structure at different densities.

As can be seen from Figures 3, 5 and 4, k-means, ROCK and CURE perform relatively poorly on these difficult data sets due to the complex shape of the clusters and the large amount of noise. DBSCAN, CHAMELEON and *TURN** work well. WaveCluster, after much tweaking of its settings, came close to finding the visually obvious clusters. DBSCAN proved very sensitive to the parameter settings. CHAMELEON requires the setting of the number of clusters to be sought, which is generally not known. It also fails to separate out noise. Combined with its high complexity , this makes it a weak contender.

*TURN** provides fast, efficient, scaleable clustering and identifies outliers allowing for the optional removal of noise as has been done in the *TURN** output presented in Figure 2. This would be useful in many applications such as OCR preprocessing, image enhancement, etc. It can stop at or flag interesting levels of granularity identified by the behaviour of global clustering features as discussed.

Here we show our experimental results of each algorithm on the DS4 data set from the CHAMELEON paper [7], also known as t7.10k. This data set was chosen because it has several features which challenge a clustering algorithm. It has nine clusters of different shapes, sizes and orientations,

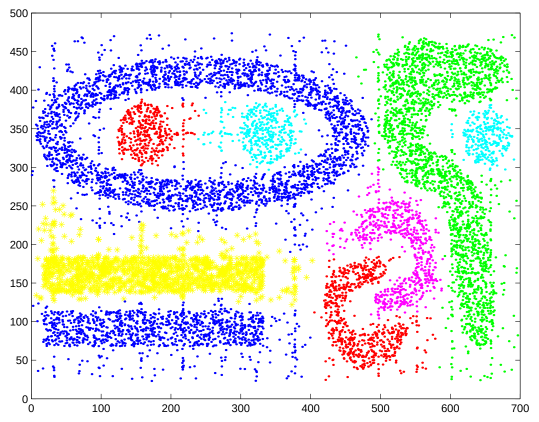

Figure 6. CHAMELEON's clustering result on t7.10k.dat with $nb\_clossest\_neighbor = 10$, MinSize $= 2.5\%$ and $k = 9$

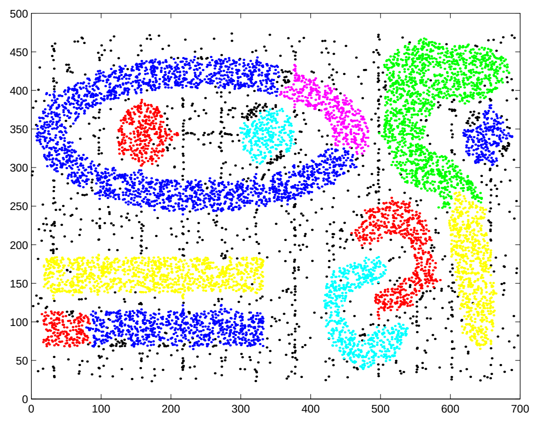

Figure 7. DBSCAN's clustering result on t7.10k.dat with $\epsilon = 5.5$ and MinPts $= 4$

and the density within and between the clusters varies. In addition, there are clusters within clusters, non-spherical shapes and a large amount of random noise which could create artificial "bridges" between the clusters. In all the clustering result figures, black points indicate data points identified by the algorithm as noise.

4.1. Clustering Effectiveness Comparison

Clustering results of *TURN** are shown in Figure 1. *TURN** correctly identified the nine principal clusters as shown in Figure 2 This cluster result shows that *TURN** can effectively identify all the 9 clusters and filter out noise. This result was found by *TURN**'s automatic resolution scan process and did not require any parameter tuning.

K-mean's result is shown in Figure 3. From here we see that k-means tends to find spherical clusters. It is obvious

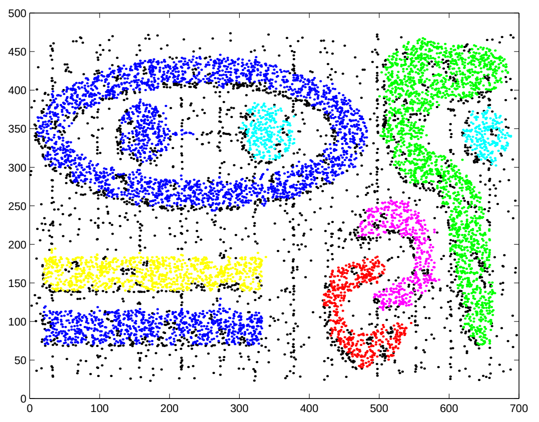

Figure 8. WaveCluster's clustering result on t7.10k.dat with $resolution = 5$ and $\tau = 1.5$

that it is not well suited to find arbitrary shaped clusters. Our experiments showed that CURE has similar problems to k-means: it tends to find spherical clusters (Figure 4). ROCK is designed for clustering categorical data but can, in theory, handle numerical data like the $t7.10k.dat$ data set. Its result for clustering this spatial data set is, however, not good. After adjusting the parameters for a long time, the best clustering result found is presented in Figure 5. For this result, we set the number of clusters to be 1000, and among the resulting 1000 clusters, we got five large clusters. The remaining 995 can be considered as noise and they all group around the upper crescent in Figure 5.

Algorithm	**Clustering time (secs)**	**Complexity**	**Memory Usage**
K_means	8.44	$O(N)$	5.5MB
CURE	155.59	$\geq O(N^2)$	4.6MB
ROCK	526.19	$> O(N^2)$	1.145GB
CHAMELEON	1667.86	$> O(NlogN)$	8.6MB
DBSCAN	10.53	$O(NlogN)$	1.4MB
WaveCluster	0.82	$O(N)$	0.8MB
TURN-RES	0.26	$O(NlogN)$	1.4MB
*TURN**	3.90	$O(NlogN)$	1.4MB

Table 1. Clustering Speed and Memory Size Results upon a 10K data set

CHAMELEON's result is shown in Figure 6, which is very close to the result in the CHAMELEON paper. Compared with *TURN**'s result, we see that CHAMELEON includes all the noise points in the neighbouring clusters. In addition, we needed to set several parameters for CHAMELEON to obtain this quality of clustering.

DBSCAN could give a good result if the adequate parameters are known. We found that the only problem of

Data Set Size	Clustering time (seconds)
10,000	0.26
100,000	3.70
228,828	8.29
275,429	9.15
574,499	22.18

Table 2. Average clustering speed of TURN-RES on one resolution across dataset sizes

DBSCAN is that it is indeed very sensitive to the two parameters ϵ and $MinPts$. Figure 7 shows the results of DBSCAN with ϵ set at 5.5 and $MinPts$ at 4. At $\epsilon = 5.9$ the result is similar to *TURN**'s but any small change in ϵ or $MinPts$ causes splitting or merging of clusters. WaveCluster's best result on t7.10k.dat was with the signal threshold on the transformed frequency domain $\tau = 1.5$ and $resolution = 5$ (Figure 8). Two clusters are joined due to the strength of the bridge in the averaged signal output. Adjusting τ (for example) resolves this problem but breaks other genuine clusters.

*TURN** was applied on both the CHAMELEON data sets [7] and the large data sets available from the WaveCluster authors (100K - 575K points) [13], and the test results are shown in Table 2 showing that the algorithm scales nearly linearly with the data set size.

5. Conclusion

The efficiency and effectiveness of clustering algorithms keeps improving. Our research confirms the weakness in the older methods and the relative benefits of the more recent algorithms. While OPTICS [9], WaveCluster [12], and other algorithms provide information at different resolution levels through dendograms or related graphs, it is left to the user to find what resolution to choose. Also, all the algorithms have certain choices - parameters - to set requiring at least some domain knowledge which is often not available to the user. Note that while our approach is non-prarametric and fully unsupervised, the user can still opt for a given resolution if desired.

In this paper, we have proposed a new clustering method called *TURN** that can build clustering information for a data set across resolutions including the equivalent of the dendograms built by other algorithms. The TURN algorithm allows us to automatically identify and, if desired, stop on the important resolution level(s) for clustering. An extension, Fuzzy TURN, not discussed here due to limits of space, permits flattening of the dendogram to identify clusters containing areas of differing densities. Together, we have a complete clustering solution which can operate unsupervised. It is fast, scales well with increasing data size, discovers clusters of arbitrary shape and is free of input parameters. It is well suited to a parallel implementation making it even faster with a near linear speedup. Our solution can also scale to higher dimensions, as will be reported elsewhere, but here we emphasise the utility in the area of two-dimensional spatial clustering. For the sake of argument and visual validation, we have used, as did other authors of clustering algorithms, two-dimensional datasets to illustrate effectiveness.

References

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

[2] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, 1996.

[3] A. Foss, W. Wang, and O. R. Zaïane. A non-parametric approach to web log analysis. In *Web Mining Workshop in conjunction with the SIAM International Conference on Data Mining*, pages 41–50, Chicago, IL, USA, April 2001.

[4] S. Guha, R. Rastogi, and K. Shim. ROCK: a robust clustering algorithm for categorical attributes. In *15th Int'l Conf. on Data Eng.*, 1999.

[5] J. Han and M. Kamber. *Data Mining, Concepts and Techniques*. Morgan Kaufmann, 2001.

[6] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[7] G. Karypis, E.-H. Han, and V. Kumar. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*, 32(8):68–75, August 1999.

[8] M. V. M. Halkidi, Y. Batistakis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, December 2001.

[9] M.Ankerst, M.Breunig, H.-P. Kriegel, and J.Sander. OPTICS: Ordering points to identify the clustering structure. In *Proc. 1999 ACM-SIGMOD Conf. on Management of Data (SIGMOD'99)*, pages 49–60, 1999.

[10] T. Masters. *Neural, Novel & Hybrid Algorithms for Time Series Prediction*. John Wiley and Sons, 1995.

[11] K. S. S. Guha, R. Rastogi. CURE: An efficient clustering algorithm for large databases. In *SIGMOD'98*, Seattle, Washington, 1998.

[12] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: a multi-resolution clustering approach for very large spatial databases. In *24th VLDB Conference*, New York, USA, 1998.

[13] G. Sheikholeslami, S. Chatterjee, and A. Zhang. A wavelet-based clustering approach for spatial data in very large databases. *The International Journal on Very Large Databases*, 8(4):289–304, February 2000.