A Dynamic Linkage Clustering using KD-Tree

Shadi Abudalfa[1] and Mohammad Mikki[2]
[1]Department of Information Technology, University Collage of Applied Sciences, Palestine
[2]Department of Computer Engineering, Islamic University of Gaza, Palestine

Abstract: *Some clustering algorithms calculate connectivity of each data point to its cluster by depending on density reachability. These algorithms can find arbitrarily shaped clusters, but they require parameters that are mostly sensitive to clustering performance. We develop a new dynamic linkage clustering algorithm using kd-tree. The proposed algorithm does not require any parameters and does not have a worst-case bound on running time that exists in many similar algorithms in the literature. Experimental results are shown in this paper to demonstrate the effectiveness of the proposed algorithm. We compare the proposed algorithm with other famous similar algorithm that is shown in literature. We present the proposed algorithm and its performance in detail along with promising avenues of future research.*

Keywords: *Data clustering, density-based clustering algorithm, KD-tree, dynamic linkage clustering, DBSCAN.*

Received February 26, 2011; accepted July 28, 2011; published online March 1, 2012

1. Introduction

Clustering [5] is a division of data into groups of similar objects. Each group, called cluster, consists of objects that are similar within the cluster and dissimilar to objects of other clusters. Representing data by fewer clusters necessarily loses certain fine details, but achieves simplification, and so may be considered as a form of data compression. It represents many data objects by few clusters models data by its clusters. Data modelling puts clustering in a historical perspective which is rooted in mathematics, statistics, and numerical analysis. Clustering is the subject of active research in several fields such as statistics, pattern recognition, artificial intelligence, and machine learning. From a practical perspective, clustering plays an outstanding role in data mining applications such as scientific data exploration, information retrieval and text mining, spatial database applications, Web analysis, marketing, medical diagnostics, computational biology, and many others.

Although, classification [6] is an effective means for distinguishing groups or classes of objects, it requires the often costly collection and labelling of a large set of training tuples or patterns, which the classifier uses to model each group. It is often more desirable to proceed in the reverse direction: First partition the set of data into groups based on data similarity (e.g., using clustering), and then assign labels to the relatively small number of groups.

From a machine learning perspective clusters correspond to hidden patterns, the search for clusters is unsupervised learning [15], and the resulting system represents a data concept. Therefore, clustering is unsupervised learning of a hidden data concept. Clustering is a challenging field of research in which its potential applications pose their own special requirements. There are many requirements of clustering in data mining such as: type of attributes algorithm can handle, scalability to large data sets, ability to work with high dimensional data [8, 9], ability to find clusters of irregular shape, handling outliers (noise), time complexity, data order dependency, labelling or assignment (hard or strict vs. soft or fuzzy [14, 16]), reliance on a priori knowledge and user defined parameters, and interpretability of results.

However, clustering is a difficult problem combinatorially, and differences in assumptions and contexts in different communities have made the transfer of useful generic concepts and methodologies slow to occur.

There are thousands of clustering techniques one can encounter in the literature. Most of the existing data clustering algorithms can be classified as hierarchical or partitional. Within each class, there exists a wealth of sub-class which includes different algorithms for finding the clusters.

While hierarchical algorithms [2] build clusters gradually (as crystals are grown), partitioning algorithms [7] learn clusters directly. In doing so, they either try to discover clusters by iteratively relocating points between subsets, or try to identify clusters as areas highly populated with data [1].

Density based algorithms [10] typically regard clusters as dense regions of objects in the data space that are separated by regions of low density. The main idea of density-based approach is to find regions of high density and low density, with high-density regions being separated from low-density regions. These approaches can make it easy to discover arbitrary clusters.

Recently, a number of clustering algorithms have been presented for spatial data, known as grid-based algorithms. They perform space segmentation and then aggregate appropriate segments [11]. Many other clustering techniques are developed, primarily in machine learning, that either have theoretical significance, are used traditionally outside the data mining community, or do not fit in previously outlined categories.

This paper deals with clustering algorithms which calculate connectivity of each data point to its cluster by depending on density reachability. Each cluster, which is a subset of the points of the data set, satisfies two properties: all points within the cluster are mutually density-connected, and if a point is density-connected to any point of the cluster, it is part of the cluster as well. These algorithms can find arbitrarily shaped clusters, but they require parameters that are mostly sensitive to clustering performance. From other side, these algorithms need to detect nearest neighborhood of each data point which cause time consuming.

We tackled with this defect and developed an original algorithm by using KD-Tree. The proposed algorithm depends on density reachability and groups data points in hierarchically way. Experimental results are shown in this paper to demonstrate the effectiveness of the proposed algorithm. We compared the proposed algorithm with other similar famous algorithm that is shown in literature. We present the proposed algorithm and its performance in detail along with promising avenues of future research.

The rest of the paper is organized as follows: section 2 describes review of literature and related studies. This section presents KD-Tree which is the most important multidimensional structure for storing a finite set of data points from k-dimensional space. Section 3 illustrates our proposed original algorithm for classifying complex data sets. We called the proposed algorithm a Dynamic Linkage Clustering using KD-Tree (DLCKDT). Section 4 illustrates experimental results to demonstrate the effectiveness of the proposed algorithm. We used synthetic and real data sets for testing efficiency of the proposed algorithm. Finally, section 5 concludes the paper and presents suggestions for future work.

2. Background

We present in this section the KD-Tree which is the most important multidimensional structure for storing a finite set of data points from k-dimensional space. In addition, the section illustrates the usage of KD-Tree. We use KD-Tree for improving performance of clustering algorithms and developing a new effective clustering algorithm. We also present Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm for comparing it with the proposed algorithm.

A K-dimensional tree, or KD-Tree [13] is a space-partitioning data structure for organizing points in a K-dimensional space. The KD-Tree is a top-down hierarchical scheme for partitioning data. Consider a set of n points, $(x_1 \ldots x_n)$ occupying an m dimensional space each point x_i has associated with it m coordinates $(x_{i1}, x_{i2}, \ldots, x_{im})$. There exists a bounding box, or bucket, which contains all data points and whose extrema are defined by the maximum and minimum coordinate values of the data points in each dimension. The data is then partitioned into two sub-buckets by splitting the data along the longest dimension of the parent bucket. These partitioning processes may then be recursively repeated on each sub-bucket until a leaf bucket is created, at which point no further partitioning will be performed on that bucket. A leaf bucket is a bucket which fulfils a certain requirement, such as, it only contains one data point.

KD-Tree is the most important multidimensional structure for storing a finite set of data points from k-dimensional space. It decomposes a multidimensional space into hyper-rectangles. A KD-Tree is a binary tree with both a dimension number and splitting value at each node. Each node corresponds to a hyper-rectangle. A hyper-rectangle is represented by an array of minimum coordinates and an array of maximum coordinates (e.g., in 2 dimensions (k=2), ($x_{\min}$, $y_{\min}$) and ($x_{\max}$, $y_{\max}$)). When searching for the nearest neighbour we need to know if a hyper-rectangle intersects with a hyper-sphere. The contents of each node are depicted in Table 1.

Table 1. The fields of KD-Tree node.

Field	Description
Type	Type of node tree (node or leaf)
Parent	The index of parent node in kd-tree
Splitdim	The splitting dimension number
Splitval	The splitting value
Left kd-Tree	A kd-tree representing points to the left of the splitting plane
Right kd-Tree	A kd-tree representing points to the right of the splitting plane
Hyperrect	The coordinates of hyperrectangle
Numpoints	The number of points contained in hyperrectangle

An interesting property of the KD-Tree is that each bucket will contain roughly the same number of points. However, if the data in a bucket is more densely packed than some other bucket we would generally expect the volume of that densely packed bucket to be smaller. Approximate Nearest Neighbour (ANN) is a library written in C++ [12], which supports data structures and algorithms for both exact and approximate nearest neighbour searching in arbitrarily high dimensions. The ANN library implements KD-Tree data structure. The function performing the k-nearest neighbor search in ANN is given a query point

q, a nonnegative integer k, an array of point indices, nn_{idx}, and an array of distances, dists. Both arrays are assumed to contain at least k elements. This procedure computes the k nearest neighbours of q in the point set and stores the indices of the nearest neighbours in the array nn_{idx}.

Each node splits the space into two subspaces according to the splitting dimension of the node, and the node's splitting value. Geometrically this represents a hyper-plane perpendicular to the direction specified by the splitting dimension. Searching for a point in the data set that is represented in a KD-Tree is accomplished in a traversal of the tree from root to leaf which is of complexity O(log(n)) (if there are n data points). The first approximation is initially found at the leaf node which contains the target point.

Our proposed algorithm is a density-based clustering algorithm because it finds a number of clusters starting from the estimated density distribution of corresponding nodes. So we compare performance of our proposed algorithm with DBSCAN algorithm. DBSCAN is a data clustering algorithm proposed by Ester *et al.* [4]. It is a density-based clustering algorithm because it finds a number of clusters starting from the estimated density distribution of corresponding nodes. DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.

DBSCAN uses two global parameters: epsilon which is used to determine the maximum radius of the point neighborhood, and minPoints which is used to determine the minimum number of points in an *Eps*-neighbourhood of that point. Performance of DBSCAN is sensitive to values of these parameters.

3. Methodology

In this section we illustrate our original work for improving efficiency of classification and tackling the problem which is presented in section 1. This section illustrates usage of KD-Tree for developing original algorithm to classify complex data sets. We called the proposed algorithm a DLCKDT. We used selected nodes from KD-Tree to develop this algorithm.

3.1. Selecting Dense Points

We proposed to use KD-Tree for checking the connectivity of each data point with its cluster. We used KD-Tree to determine the collections of dense regions in dimensional space. Using KD-Tree will reduce computation cost and its results will be better than using other methods that are presented in literature for determining the dense regions. We selected some points of KD-Tree which denote the dense centers of dense regions in the data set. We called these points as Dense Points (DPs).

Selecting leaf nodes as DPs is not suitable because each leaf node in KD-Tree is a bucket contains only one data point and will cause selecting all data points in the data set. So selecting leaf nodes as DPs will not form dense centers of dense regions in the data set.

Selecting parent of leaf nodes in KD-Tree as DPs is not suitable also because parent of leaf node contains only two data points (two leaf nodes) and will cause sensitivity to noise (outlying data points) in the data set.

Depending on the previous analysis, we selected DPs by searching for leaf nodes in the KD-Tree and then finding the grandparent of the leaf nodes. Grandparent of the leaf nodes contains more than two data points, so selecting them as DPs will reduce sensitivity to noise in the data set and will form small number of centers to denote to dense regions in the data set for reducing processing time in classification. Figure 1 shows the structure of KD-Tree and the position of DPs. We note that the DPs (which are shown as shaded nodes) denote to 2nd and 3rd levels of the KD-Tree. We note that more than two nodes fork from nodes of DPs, this indicates to DPs contains more than two data points.

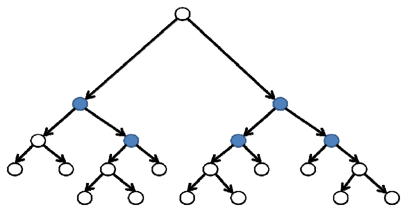


Figure 1. Selected dense points.

Figure 2-a shows the position of DPs on synthetic data set which has one cluster. We note that these points form almost the shape of cluster with little number of points. We note that DPs distributed on the whole data set and they exist in dense regions of data points. Figure 2-b shows the rectangular regions which are covered by DPs of KD-Tree. We note that these regions almost cover all the data set, so we can conclude that DPs correspond to the dense regions of the data set.

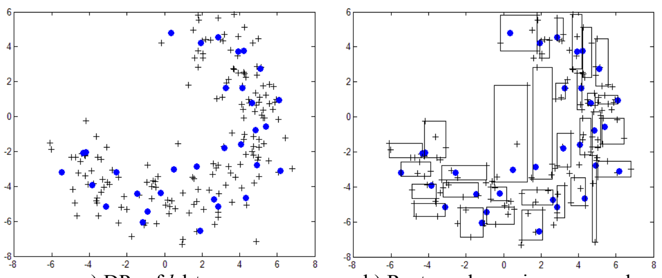
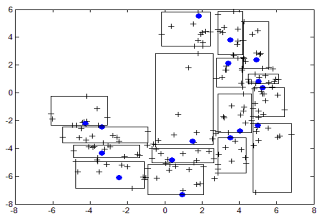

a) DPs of *k*d-tree. b) Rectangular regions covered by DPs.

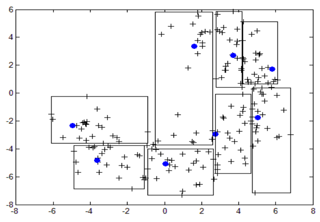
Figure 2. Illustrating how DPs cover all the data set.

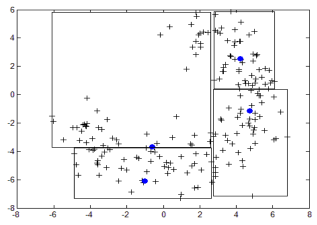
Using upper levels in KD-Tree (more than the 3rd level) for selecting DPs will decrease number of DPs for representing dense regions, but in the same time rectangular regions will be larger and will cover some parts of space which are empty from data points. Figure 3 shows selecting nodes from various levels in KD-Tree and showing the corresponding of these nodes to data points in a data set of one cluster.

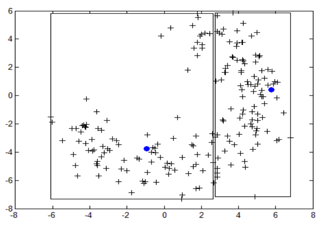
We note from Figure 3-a that the number of nodes which denote to dense regions are smaller than number of DPs which are shown in Figure 2-b but the size of rectangular regions are increased, this caused covering empty regions of data points. These effects are increased gradualness from Figure 3-c to Figure 3-e. Figure 3-e shows that only one node represents all data points in cluster and covers empty space outside the cluster. So we inferred, if we use upper levels for representing DPs then the shape which is formed by rectangular regions for covering the cluster will be rough, and many data points will be selected from other clusters if there are overlapped clusters in the data set.

We can conclude that selecting the grandparent of the leaf nodes in KD-Tree for representing DPs is the best choice to determine the collections of dense regions in dimensional space. We used this concept for selecting DPs in our experiments for increasing performance of classifying clusters in complex data set.


a) Nodes of the 4th level.


b) Nodes of the 5th level.


c) Nodes of the 6th level.


d) Nodes of the 7th level.

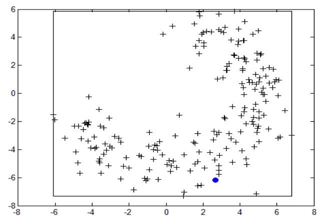

e) Nodes of the 8th level.

Figure 3. Selecting nodes from various levels in KD-Tree upper than the 3rd level.

Selecting DPs have many advantages. First of all, using DPs reduces the number of data points used for classification, so this method will reduce time complexity. From other side, using DPs will reduce the effect of noise (outlying data points) on classification. Figure 4 shows position of DPs (plotted as circles) in data set having one cluster with outlying data points. We note that the outlying data points, which denoted as +symbols in the four corners of the figure, is not selected as DPs. We note also that all DPs are concentrated in spaces which have density of data points.

So we can use DPs for checking density reachability of each data point with its cluster. Using DPs will be effective for classifying complex data sets which have overlapped and arbitrary shaped clusters.

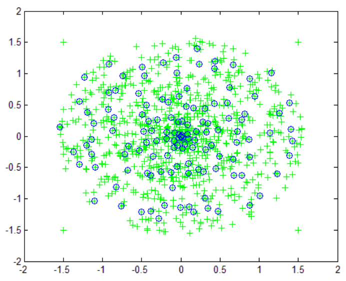


Figure 4. Selecting DPs form data set having noise.

Next, we will use DPs for improving efficacy of clustering algorithms and developing a new effective clustering algorithm.

3.2. Dynamic Linkage Clustering using KD-Tree

In this section we develop a new clustering algorithm depending on the KD-Tree. We called the proposed algorithm as a DLCKDT. We used selected nodes from KD-Tree to develop this algorithm. Our goal for developing this algorithm is classifying complex data sets more accurately than other algorithms which are presented in the literature.

The new developed clustering algorithm depends on the KD-Tree. It consists of three phases: The first phase selects DPs of KD-Tree for using them as initial seeds. The second phase assigns each data set to its nearest DP. So the output of this phase is a collection of small clusters whose number is equal to the number of DPs. The last phase merges the small clusters (output of the second phase). During the third phase, each iteration consists of merging the nearest two clusters. This phase continues until the number of clusters is equal to the value which is specified by the user in advance (denoted by the number of classes).
The pseudo code of our novel clustering algorithm using KD-Tree is:

```
1. Input the number of clusters K.
2. Select DPs of KD-Tree ($DP_1$,..., $DP_n$).
3. Assign each data point $X_i$ to its nearest $DP_j$ to form
   initial clusters of data points.
4. Merge every two adjacent clusters of step 3.
5. Find the nearest two clusters and merge them.
6. If the number of merged clusters $N>K$, then go to
   step 5; else return the merged clusters.
```

Step 2 is the first phase in the algorithm for selecting DPs of KD-Tree. Step 3 is the second phase. It creates a large number of small groups which are used as initial clusters. Steps 4, 5, and 6 form the final phase. This phase merges clusters which are generated in step 3. The merging is terminated when the number of merged clusters is equal to the value of K, where K is an input parameter which is defined as the target number of clusters. Step 4 decreases to half the number of selected clusters which are generated by step 3. This step is used for reducing time complexity. We used the nearest neighbor distance [17] to calculate the distance between each two clusters C_1 and C_2 which is denoted by where:

$$D(C_1, C_2)=\min_{1\le i\le r, 1\le j\le s} d(y_i, z_i) \tag{1}$$

Figure 5 gives an example of the nearest neighbor distance in the two-dimensional case. We note that $D(C_1, C_2)$ is the Euclidean distance between the nearest points between clusters C_1 and C_2. For calculating this equation we need to calculate all distances between each point in C_1 and C_2 and then finding the minimum distance. We can find the nearest two clusters for step 5 of the algorithm by calculating the minimum distance between all clusters.

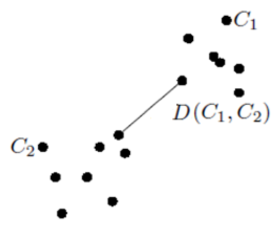


Figure 5. Nearest neighbour distance between two clusters.

DLCKDT has three main advantages:

1. It is easy to implement.
2. The algorithm does not depend on the initial conditions. This forces the algorithm to converge to global solution.
3. It can classify very complex data sets which cannot be classified by other clustering algorithms in the literature.

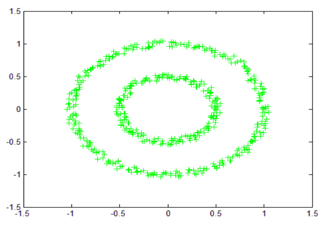
However the proposed algorithm suffers from the following disadvantages:

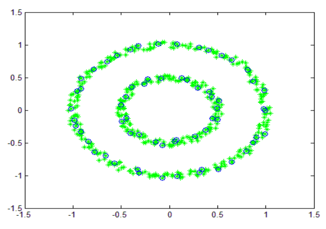
1. The user has to specify the number of classes in advance.
2. The elapsed time is increased when comparing it with other clustering algorithms in the literature.

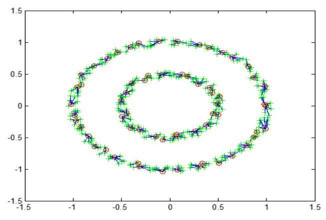
4. Experimental Results

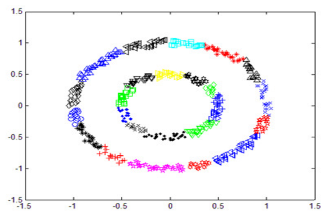
Experimental results are shown in this section to demonstrate the effectiveness of the DLCKDT. We implemented DLCKDT algorithm by using MATLAB 7.3 (R2006b) for illustrating its performance. We used synthetic and real data sets for testing efficiency of the proposed algorithm.

We used our algorithm for classifying a lot of complex data sets. Figure 6 illustrates the power of our algorithm. We used a data set that consists of two circles (two clusters); one of them is inside the other as shown in Figure 6-a. Figure 6-b shows the DPs (marked as circles) of KD-Tree. We note that DPs are distributed though whole data set, and located in the dense regions. We note that the DPs formed the shape of clusters with small number of data points. Figure 6-c shows the clusters of DPs. We note that every data point is connected to the nearest DP. We note that this step generates a collection of small clusters where DPs form their centroids. We note that the number of these groups equal to number of used DPs. The results of merging every two adjacent clusters of DPs are shown in Figure 6-d. We note that the number of total groups, which are marked with different colors and shapes, is reduced to half (only 24 groups). The output of the algorithm is shown in Figure 6-e. We note that data set is classified correctly into to clusters. Data points of the first cluster (the smallest circle) are marked as circles and the data points of the second cluster (the largest circle) are marked as triangles.


a) Complex Synthetic data set.


b) DPs of *k*d-tree.


c) Clusters of DPs.


d) Merging every two adjacent clusters of DPs.

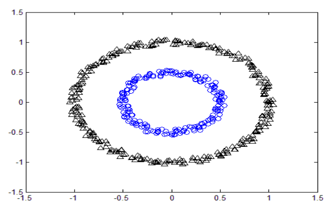

e) The output of algorithm.

Figure 6. Effectiveness of the proposed algorithm.

The following real-life data sets [3] are used for testing performance of our proposed algorithm DLCKDT and DBSCAN. Here, n is the number of data points, d is the number of features, and K is the number of clusters.

1. *Iris Plants Data Set (n=150, d=4, K=3):* This is a well-known data set with 4 inputs, 3 classes, and

150 data vectors. The data set consists of three different species of iris flower: Iris setosa, Iris virginica, and Iris versicolour. One class is linearly separable from the other 2; the latter are not linearly separable from each other. For each species, 50 samples with four features each (sepal length, sepal width, petal length, and petal width) were collected. The number of objects that belong to each cluster is 50.

2. *Abalone Data Set (n=1253, d=8, K=3):* This is a data set for predicting the age of abalone from physical measurements. The data were sampled from three clusters: the first cluster has 397objects, the second cluster has 434objects, and the last cluster has 422objects. The data contains eight relevant features: 1). sex, 2). length, 3). diameter, 4). height, 5). whole weight, 6). shucked weight, 7). viscera weight, and 8). shell weight.
3. *Contraceptive Method Choice Data Set (n=1473, d=9, K=3):* This data set is a subset of the 1987 National Indonesia Contraceptive Prevalence Survey. The samples are married women who were either not pregnant or do not know if they were at the time of interview. The problem is to predict the current contraceptive method choice: 1). no use (629objects), 2). long-term methods (333objects), or 3). short-term methods (511objects) of a woman based on her demographic and socio-economic characteristics. The data contains nine relevant features: 1). wife's age, 2). wife's education, 3). wife's education, 4). number of children ever born, 5). wife's religion, 6). wife's now working?, 7). husband's occupation, 8). standard-of-living index, and 9). media exposure.
4. *Haberman's Survival Data Set (n=306, d=3, K=2):* The data set contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer. The objective is to classify each data vector into: 1). the patient survived 5years or longer (225objects), or 2). the patient died within 5year (81objects). The data contains three relevant features: 1). age of patient at time of operation, 2). patient's year of operation, and 3). number of positive axillary nodes detected.
5. *Heart Disease Data Set (n=303, d=13, K=2):* This is a data set with 13 inputs, 2 classes, and 303 data vectors. The "goal" field refers to the presence of heart disease in the patient. The problem is to predict the diagnosis of heart disease (angiographic disease status) by classifying each data vector into: 1)<50% diameter narrowing (164objects); or 2)> 50% diameter narrowing (139objects). The data contains 13 relevant features: 1). age, 2). sex, 3). chest pain type, 4). resting blood pressure, 5). serum cholestoral, 6). fasting blood sugar>120 mg/dl?, 7). resting electrocardiographic results, 8). maximum heart rate achieved, 9). exercise induced angina?, 10). ST depression induced by exercise relative to rest, 11). the slope of the peak exercise ST segment, 12). number of major vessels, and 13). that (normal, fixed defect, or reversable defect).

We used Waikato Environment for Knowledge Analysis (WEKA) for classifying data sets by DBSCAN algorithm. WEKA is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. WEKA contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.

We assigned input parameters of DBSCAN algorithm to epsilon=0.9 and minPoints=6 for classifying all data sets. We tested performance of DBSCAN and our algorithm DLCKDT by counting the data points which are classified incorrectly. The data set description and the individual performance of DBSCAN algorithm and our algorithm DLCKDT are summarized in Table 2.

Table 2. The data sets description, percentage of incorrectly clustered instances by DBSCAN algorithm and DLCKDT algorithm.

#	Data set Name	n	D	K	Percentage of Incorrectly Clustered Instances by Using DBSCAN (%)	Percentage of Incorrectly Clustered Instances by Using DLCKDT (%)
1	Iris	150	4	3	66.67	26
2	Abalone	1253	8	3	65.36	49.16
3	Contraceptive method choice	1473	9	3	59.47	57.77
4	Haberman's survival	306	3	2	87.91	25.82
5	Heart disease	303	13	2	59.08	43.89

We observed that the proposed algorithm performed very well. We found that the DBSCAN algorithm failed to classify 68% of the average number of all instances in the data sets while DLCKDT performed 41%. We can conclude that the proposed algorithm performs better performance than DBSCAN algorithm and it doesn’t reliance on a priori knowledge and user defined parameters like DBSCAN.

5. Conclusions and Future Work

In this paper, we described an essential problem in data clustering and presented some solutions for it. We developed a novel clustering algorithm by using *k*d-tree and we proved its performance. The proposed algorithm did not have a worst-case bound on running time. Experimental results are shown in this paper to demonstrate the effectiveness of the proposed algorithm. We illustrated the performance of classifying complex data sets. We proved that the proposed algorithm can classify complex data sets more accurately than other algorithms presented in the literature. The work reported in this paper may be extended in a number of ways, some of which are discussed below:

1. We used KD-Tree for improving the performance of classification. Many optimizing search strategies in KD-Tree are developed in literature. We can use these strategies for improving the time complexity of our algorithm and study its performance.
2. Our proposed algorithm depends on KD-Tree for improving the performance of clustering. It is interesting to study some other kinds of trees like R+_tree and Bkd-tree: A Dynamic Scalable KD-Tree.

References

[1] Abu-Abbas O., "Comparisons Between Data Clustering Algorithms," *The International Arab Journal of Information Technology*, vol. 5, no. 3, pp. 320-325, 2008.

[2] Agarwal P., Alam M., and Biswas R., "Analysing the Agglomerative Hierarchical Clustering Algorithm for Categorical Attributes," *International Journal of Innovation, Management and Technology*, vol. 1, no. 2, pp. 186-190, 2010.

[3] Asuncion A. and Newman D., "UCI Repository of Machine Learning Database," available at: http://www.ics.uci.edu/~mlearn/MLrepository.html, last visited 2007.

[4] Ester M., Kriegel H., Sander J., and Xu X., "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *in Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, Portland, pp. 226-231,1996.

[5] Gan G., Ma C., and Wu J., *Data Clustering: Theory, Algorithms, and Applications*, ASA-SIAM Series on Statistics and Applied Probability, SIAM, Philadelphia, Alexandria, 2007.

[6] Halgamuge S. and Wang L., *Classification and Clustering for Knowledge Discover*, Springer-Verlag Berlin Heidelberg, New York, 2005.

[7] Hammerly G. and Elkan C., "Alternatives to the k-Means Algorithm That Find Better Clusterings," *in Proceedings of the 11th International Conference on Information and Knowledge Management*, USA, pp. 600-607, 2002.

[8] Kogan J., *Introduction to Clustering Large and High-Dimensional Data*, Cambridge University Press, New York, 2007.

[9] Kogan J., Teboulle M., and Nicholas C., *Grouping Multidimensional Data*, Springer-Verlag, New York, 2006.

[10] Kriegel H., Kröger P., Sander J., and Zimek A., "Density-Based Clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231-240, 2011.

[11] Lin N., Chang C., Jan N., Chen H., and Hao W., "A Deflected Grid-Based Algorithm for Clustering Analysis," *International Journal of Mathematical Models and Methods in Applied Sciences*, vol. 1, no. 1, pp. 33-39, 2007.

[12] Mount D. and Arya S., "ANN: A Library for Approximate Nearest Neighbor Searching," available at: http://www.cs.umd.edu/~mount/ANN, last visited 2005.

[13] Panigrahy R., "An Improved Algorithm Finding Nearest Neighbor Using KD-Trees," *in Proceedings of the 8th Latin American Conference on Theoretical Informatics*, Berlin, pp. 387-398, 2008.

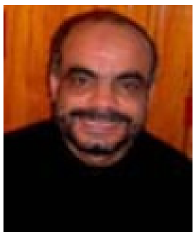
[14] Sato-Ilic M. and Jain L., *Innovations in Fuzzy Clustering*, Springer-Verlag, New York, 2006.

[15] Theodoridis S. and Koutroumbas K., *Pattern Recognition*, Elsevier Academic Press, Amsterdam, 2003.

[16] Valente J. and Pedrycz W., *Advances in Fuzzy Clustering and its Applications*, John Wiley & Sons Ltd, England, 2007.

[17] Williams W. and Lambert J., "Multivariate Methods in Plant Ecology: V. Similarity Analyses and Information-Analysis," *Journal of Ecology*, vol. 54, no. 2, pp. 427-445, 1966.

Shadi Abudalfa received his BSc and MSc degrees both in computer engineering from the Islamic University of Gaza, Palestine in 2003 and 2010 respectively. He is a lecturer at the University Collage of Applied Sciences, Palestine. From July 2003 to August 2004, he worked as a research assistant at projects and research Lab in IUG. From February 2004 to August 2004, he worked as a teaching assistant at Faculty of Engineering in IUG.



Mohammad Mikki is a professor of computer engineering at Islamic University of Gaza, Palestine. His general research interests are in high performance parallel and distributed computing, high speed computer networks and communications protocols and computer networks management, modeling and design of digital computer systems, internet technology and programming, internet performance measurement tools, and web-based learning.