JOURNAL OF COMPLEXITY **18**, 375–391 (2002)
doi:10.1006/jcom.2001.0633, available online at http://www.idealibrary.com on IDEAL®

The New *k*-Windows Algorithm for Improving the *k*-Means Clustering Algorithm

M. N. Vrahatis

Department of Mathematics, University of Patras (UOP), University of Patras Artificial Intelligence Research Center (UPAIRC), GR-26500 Patras, Greece
E-mail: vrahatis@math.upatras.gr

B. Boutsinas

Department of Business Administration, UOP, UPAIRC, GR-26500 Patras, Greece
E-mail: vutsinas@bma.upatras.gr

P. Alevizos

Department of Mathematics, UOP, UPAIRC, GR-26500 Patras, Greece
E-mail: alevizos@math.upatras.gr

and

G. Pavlides

Department of Computer Engineering and Inf., UOP, UPAIRC, GR-26500 Patras, Greece
E-mail: pavlidis@cti.upatras.gr

Received June 30, 2000; revised March 1, 2001; accepted October 1, 2001

The process of partitioning a large set of patterns into disjoint and homogeneous clusters is fundamental in knowledge acquisition. It is called *Clustering* in the literature and it is applied in various fields including data mining, statistical data analysis, compression and vector quantization. The *k-means* is a very popular algorithm and one of the best for implementing the clustering process. The *k*-means has a time complexity that is dominated by the product of the number of patterns, the number of clusters, and the number of iterations. Also, it often converges to a local minimum. In this paper, we present an improvement of the *k*-means clustering algorithm, aiming at a better time complexity and partitioning accuracy. Our approach reduces the number of patterns that need to be examined for similarity, in each iteration, using a windowing technique. The latter is based on well known spatial data structures, namely the range tree, that allows fast range searches. © 2002 Elsevier Science (USA)

Key Words: *k*-means clustering algorithm; unsupervised learning; data mining; range search.

1. INTRODUCTION

Recently, the task of extracting knowledge from databases has become a subject of great interest. This is mainly due to the explosive growth in the use of databases and the huge volume of data stored in them. A lot of techniques have been proposed in the literature for data mining processes (e.g., [10, 18]).

Clustering is one of the most popular data mining tasks that consists in partitioning a large set of patterns into disjoint and homogeneous clusters. Usually, clustering algorithms output the means of discovered clusters. Providing that these means are the representatives of the clusters [1], the conjunction of attribute values describing each mean can be considered as a clustering rule for describing data (taking, of course, into consideration and a number of certain properties as density, variance, shape, and separation [1]).

Clustering rules are one of the most common representation formalisms for extracted knowledge. Clustering rules can be extracted using unsupervised learning methods and can be used to partition a set of patterns into disjoint and homogeneous clusters. Clustering algorithms can be classified as either partitional clustering or hierarchical clustering algorithms and they have been widely studied in various fields including machine learning, neural networks, and statistics. Clustering algorithms can be applied in various other fields such as statistical data analysis, compression and vector quantization.

The *k-means* [14, 21], along with its variants (e.g., [3, 13, 24]), is a popular algorithm that has been used in various practical applications. However, *k*-means is computationally very expensive for the very large sets of patterns met in real life applications. On the other hand, *k*-means often converges to a local minimum.

In this paper, we present the *k*-windows algorithm which is a modification of the *k*-means clustering algorithm. The *k*-windows algorithm aims at a better time complexity and greater partitioning accuracy. Our approach reduces the number of patterns that need to be examined for similarity, in each iteration, using a windowing technique. The latter is based on well known spatial data structures, namely the range tree, that allows fast range searches.

The rest of the paper is organized as follows. The direct *k*-means algorithm, along with the types of extensions, is described briefly in Section 2. The proposed *k*-windows algorithm is described in Section 3, while its computational complexity, is given in Section 4. In Section 5 empirical tests are presented providing experimental evidence of the improvement achieved. The paper ends with some concluding remarks and a short discussion for further research.

2. THE DIRECT k-MEANS ALGORITHM

The *k-means* is a very popular algorithm particularly suited for implementing the clustering process because of its ability to effciently partition huge amounts of patterns. The latter is true even in the presence of noise. Although direct k-means is defined over numerical continuous data, it is the basic framework for defining variants capable of working on both numerical and categorical data.

The k-means consists of two main phases. During the first phase, a partition of patterns, in k clusters is calculated, while during the second phase, the quality of the partition is determined. k-means is implemented by an iterative process that starts from a random initial partition. The latter is repeatedly recalculated until its quality function reaches an optimum. In particular, the whole process is built upon four basic steps:

(1) selection of the initial k means,

(2) assignment of each pattern to a cluster with nearest mean,

(3) recalculation of k means for clusters, and

(4) computation of the quality function.

The last three steps are performed iteratively until convergence. Most clustering algorithms which are variants of k-means have been proved convergent [26]. On the other hand, k-means-type algorithms often terminate at a local minimum.

Formally, let $i_1, \ldots, i_n$ be the input patterns. Each of them is represented by a d-tuple $\{(an_1, av_1), \ldots, (an_d, av_d)\}$ where $an_j, av_j, 1 \leqslant j \leqslant d$ denote, respectively, the name and the value of the jth numerical attribute, whose domain is the set of reals $\mathbb{R}$. Let the k first means be initialized to one of n input patterns $i_{m1}, \ldots, i_{mk}$. These k means define the set C of clusters $C = \{C_j \mid 1 \leqslant j \leqslant k\}$. The essence of the algorithm is to minimize the following quality function:

$$E = \sum_{j=1}^{k} \sum_{i_l \in C_j} q(i_l, i_{mj}).$$

In direct k-means q is defined by the squared Euclidean distance, thus $q(x, y) = \|x - y\|^2$, where $\|\cdot\|$ determines the Euclidean norm. Therefore, the direct k-means clustering algorithm is as follows:

ALGORITHM DIRECT k-MEANS.

```
input $k$
initialize $k$ means $i_{m1}, \ldots, i_{mk}$
repeat
    for each input pattern $i_l$, $1 \leqslant l \leqslant n$
        do
            assign $i_l$ to $C_j$ with nearest mean $i_{mj}$,
            such as $\|i_l - i_{mj}\|^2 \leqslant \|i_l - i_{mu}\|^2$, $1 \leqslant j, u \leqslant k$
    for each cluster $C_j \in C$, $1 \leqslant j \leqslant k$
        do
            recalculate the mean of patterns $i_l \in C_j$, $i_{mj} = \frac{1}{|C_j|} \sum_{i_l \in C_j} i_l$
            where $|C_j|$ defines the cardinality of $C_j$
    compute the quality function $q$
until no object has changed clusters (or $q$ does not change)
```

The direct k-means algorithm is computationally very expensive for large sets of patterns. It requires time proportional to the product of the number of patterns, the number of clusters and the number of iterations. More specifically, in the algorithm above, the first loop, for each iteration, has a time complexity $O(ndk)$, the second $O(nd)$ and the quality function is calculated in $O(nd)$. Thus the whole algorithm has a time complexity $O(ndkt)$, where t is the number of iterations. In practice, it holds that $d, k, t \ll n$. Note that the first loop has as a basic operation the calculation of the squared Euclidean distance of two numbers and it is this which we consider the basic unit of computational processing cost. The calculation of the quality function has the same basic operation, while the second loop has as a basic operation just the addition of two numbers.

There are a number of modifications in the direct k-means algorithm improving either the computational complexity or the expressive adequacy. The latter is achieved by extending the direct k-means to work on categorical date (e.g., [13]) or on mixed data (e.g., [22]). Another related extension concerns the quality function, where different (dis)similarity measures have been proposed (e.g., [11, 12, 24]). Improvement of the computational complexity is achieved either by sophisticated initialization methods (e.g., [6, 13, 19]) or by reducing the number of (dis)similarity calculations (e.g., [3, 15, 23]). The k-windows algorithm is based on the latter approach.

3. THE k-WINDOWS ALGORITHM

The time complexity of the direct k-means algorithm is determined, mainly, by the number of patterns, especially when it scales to a very large set of patterns. More specifically, the step of assignment of each pattern to the cluster with the nearest mean is computationally the most expensive. This is imposed not only by its time complexity in relative terms, but, also, by its basic operation which is the calculation of the squared Euclidean distance. The latter is computationally expensive in absolute terms.

The proposed k-windows algorithm deals with this problem by using a windowing technique, which reduces significantly the number of patterns that need to be examined at each iteration. Moreover, the basic operation in the first loop, during the assignment of patterns to clusters, is now just the arithmetic comparison between two numbers.

The key idea behind the proposed technique is to use a window in order to determine a cluster. The window is defined as an orthogonal range in the d-dimensional Euclidean space, where d is the number of numerical attributes. Therefore each window is a *d-range* of an initially fixed area a. The magnitude of a depends on the density of the data set. In empirical tests presented in Section 5, we choose to define, across each different direction i,

$$a_i = \frac{(\textit{mean distance among patterns in } i)}{(\textit{number of windows})} \times 0.5.$$

Intuitively, we try to fill the mean space between two patterns with non overlapping (thus we scale by 0.5) windows. Every pattern that lies within a window is considered as belonging to the corresponding cluster. Iteratively, each window is moved in the Euclidean space by centering itself on the mean of the patterns included. This takes place until no further movement results in an increase in the number of patterns that lie within it (see solid line squares in Fig. 1). After this step, we can determine the means of

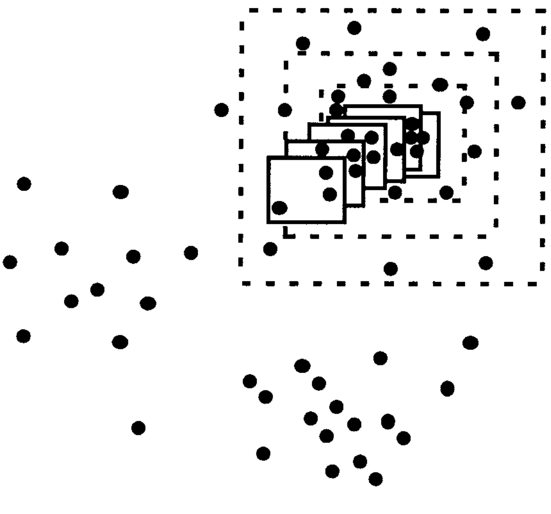

FIG. 1. Movements and enlargements of a window.

clusters as the means of the corresponding windows. However, since only a limited number of patterns is considered in each movement, the quality of a partition may not be optimum. The quality of a partition is calculated in a second phase. At first, we enlarge windows in order to contain as many patterns from the corresponding cluster as possible (see dotted line squares in Fig. 1). The quality of a partition is determined by the number of patterns contained in any window, with respect to all patterns.

The proposed k-windows clustering algorithm is as follows:

Algorithm k-Windows.

```
input $k$, $a$, $v$
initialize $k$ means $i_{m1}, \ldots, i_{mk}$ along with their
$k$ $d$-ranges $w_{m1}, \ldots, w_{mk}$ each of area $a$
repeat
    for each input pattern $i_l$, $1 \leqslant l \leqslant n$
        do
            assign $i_l$ to $w_j$ ,
            so that $i_l$ lies within $w_j$
    for each $d$-range $w_j$
        do
            calculate its mean $i_{mj} = \frac{1}{|w_j|} \sum_{i_l \in w_j} i_l$
            and recalculate $d$-ranges
until no pattern has changed $d$-ranges
enlarge $d$-ranges up to no significant
change exists, in their initial mean
compute the ratio $r = \frac{1}{n} \sum_{j=1}^{k} |i_l \in w_j|$
if $r < v$
    do
        reexecute the algorithm
```

At first, k means are selected (possibly in a random way). Initial d-ranges (windows) have as centers these initial means and each one is of area a. Then, the patterns that lie within each d-range are found. If a brute search were used, the time complexity of this step would still be determined by the number of patterns. Instead, an orthogonal range search [9, 20] is used. An orthogonal range search is based on a preprocess phase where a *range tree* is constructed (see next section). Patterns that lie within a d-range can be found by traversing the range tree, in polylogarithmic time. In the third step, the mean of patterns, that lie within each range, is calculated. Each such mean defines a new d-range, that is considered a movement of the previous d-range. The last two steps are executed repeatedly, until no d-range includes a significant increment of patterns after a movement.

In a second phase, the quality of the partition is calculated. At first, the d-ranges are enlarged in order to include as many patterns as possible from the cluster. This can be achieved by forcing d-ranges to preserve their mean during enlargement. Then, the relative frequency of patterns assigned to a d-range in the whole set of patterns, is calculated. If the relative frequency is small with respect to the user defined threshold v, then, possibly, there may be a missing cluster (or clusters) (see Fig. 2). In that case, the whole process is repeated. We are, currently, investigating various approaches that can be used to guide such repetitions. For instance, a new repetition can be started with the same initial means but with d-ranges of a larger area $a' > a$. Another approach is to start with different initial means located at a maximum distance from the previous ones. However, in almost all of the tests we have made,there was no need for reexecuting the algorithm. In conclusion, the value of v is a user defined threshold which can be used as a termination criterion of the algorithm. According to our experience, this value does not play a critical role, if the initial d-ranges have successfully been chosen. We have used it for completeness purposes.

In contrast to direct k-means, the basic operation, during the assignment of patterns, is arithmetic comparison between two numbers. Such comparisons guide the traversal of the range tree from its root to a leaf. In the following section we present the orthogonal range search along with its complexity.

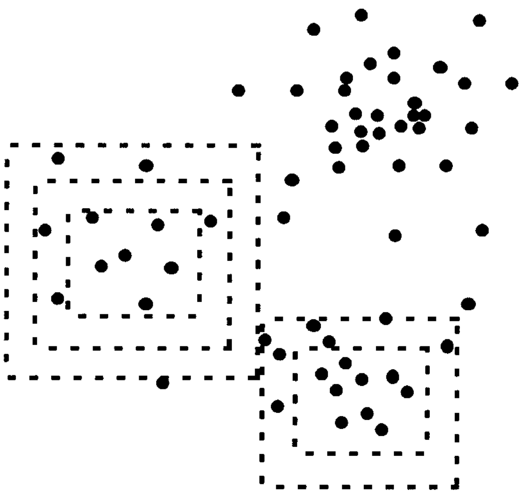

FIG. 2. Missing cluster(s).

4. ANALYSIS AND COMPUTATIONAL COMPLEXITY

To analyze the computational complexity of the proposed algorithm, we use the Orthogonal Range Search notion of computational geometry. This notion has been shown to be effective in many practical applications and a considerable amount of work has been devoted to this problem [20]. Our approach is heavily based on this notion. Thus, for completeness we briefly describe in this section the orthogonal range search problem and the techniques and data structures that are used for solving it.

The *orthogonal range search* problem can be stated as follows:

Input:
a) $V = \{p_1, \ldots, p_n\}$ is a set of n points in $\mathbb{R}^d$ the d-dimensional Euclidean space with coordinate axes $(Ox_1, \ldots, Ox_d)$,
b) a query d-range $\mathcal{Q} = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ is specified by two points $(a_1, a_2, \ldots, a_d)$ and $(b_1, b_2, \ldots, b_d)$, with $a_j \leqslant b_j$.
Output:
report all points of V that lie within the d-range $\mathcal{Q}$.

All efforts on range searching discuss how to preprocess a class of objects, namely points, for efficiently answering range search queries with specific range types. The most extensively studied type of query range is the orthogonal range.

The well-known method of the *Range Tree* allows us to solve orthogonal range queries in $O(n \log^{d-1} n)$ preprocessing time and space and $O(s + \log^d n)$ query time, if s points are retrieved. A d-dimensional range tree consists of a balanced binary leaf-search tree T which stores in its leaves the points of the given set V in increasing order with respect to their first coordinate. Any internal node p_t of T stores the point that appears in the rightmost leaf of the left subtree of p_t. To each internal node p_s of a $(d-i)$-dimensional range tree $(\forall i \in \{0, 1, \ldots, d-1\})$ is associated a $(d-i-1)$-dimensional range tree T_{p_s}, which stores in its leaves all points in the subtree rooted at p_s in increasing order with respect to their $i+1$ coordinate (see Fig. 3).

To perform a range search with the d-range $\mathcal{Q}$ we begin by searching with both a_1 and b_1 (with $a_1 < b_1$) in the d-dimensional range tree T in order to find the two leaves, let $p_a = (x_1^a, \ldots x_d^a)$ and $p_b = (x_1^b, \ldots, x_d^b)$ in T, such that p_a is the nearest before $a_1 (x_1^a < a_1)$ and p_b is the nearest after $b_1 (b_1 < x_1^b)$. For some time the search for a_1 and b_1 may follow the same path, but at some node p_t we will find that a_1 lies below the leftchild of p_t and b_1 lies below the rightchild of p_t (see Fig. 4).

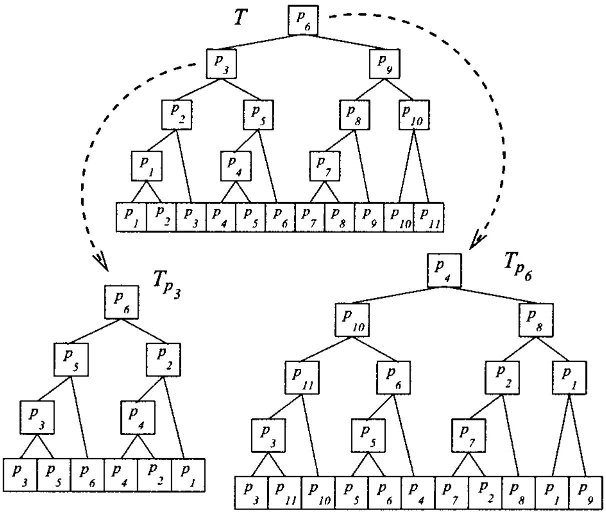
T
T_{p_3}
T_{p_6}
p_6 p_3 p_9 p_2 p_5 p_8 p_{10} p_1 p_4 p_7
p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11}
p_6 p_5 p_2 p_3 p_4
p_3 p_5 p_6 p_4 p_2 p_1
p_4 p_{10} p_8 p_{11} p_6 p_2 p_1 p_3 p_5 p_7
p_3 p_{11} p_{10} p_5 p_6 p_4 p_7 p_2 p_8 p_1 p_9

FIG. 3. The range tree of the set $V = p_1, \ldots, p_n$ where $p_i = (x_1^i; x_2^i)$ $\mathbb{R}^2$ such that $x_1^1 < x_1^2 < \cdots < x_1^{11}$ and $x_2^3 < x_2^{11} < x_2^{10} < \cdots < x_2^8 < x_2^1 < x_2^9$. In this figure only a part of the 1-dimensional range trees T_{p_i} is exhibited.

Consider the nodes $p_w \in T$ that are either rightchilds of a node on the path $p_t, \ldots, p_a$ or leftchilds of a node on the path $p_t, \ldots, p_b$. Because T is balanced, there are at most $O(\log n)$ such nodes p_w. Consequently, we search with both a_2 and b_2 in each range tree T_{p_w}. In this search, the union of the answers, over $O(\log n)(d-1)$-dimensional range trees T_{p_w}, consists of at most $O(\log^2 n)$ nodes. In conclusion, at the end of the algorithm, we search with both a_d and b_d in at most $O(\log^{d-1} n)$ 2-dimensional range trees and we will find at most $O(\log^d n)$ 1-dimensional range trees. It is obvious that all points stored in the leaves of the last trees lie within $\mathcal{Q}$.

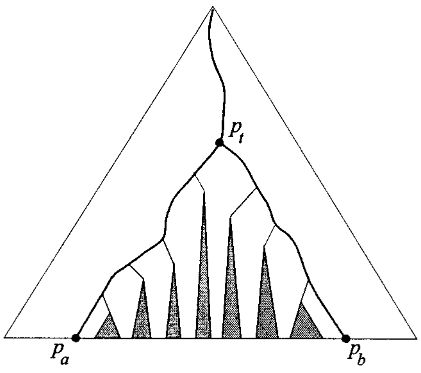

FIG. 4. The open interval $(p_a; p_b)$ in the list of leaves is partitioned in $O(\log n)$ subintervals.

We note that, based on the Wilard–Lueeker modification [20] of the range tree, known as *layered* range tree, the range searching can be performed in $O(n \log^{d-1} n)$ preprocessing time and space and $O(s+\log^{d-1} n)$ query time. In the literature there are different solutions to the orthogonal range search problem. With the *multidimensional binary tree* method [20], the answer is given in time $O(s+dn^{1-(1/d)})$ using $\theta(dn)$ storage and $\theta(dn \log n)$ preprocessing time. In [8] Chazelle introduces a new approach called *filtering search* which leads to a data structure in $O(n(\log^{d-1} n/\log \log n))$ space and $O(n \log^{d-1} n)$ time for answering in time $O(s+\log^{d-1} n)$. For orthogonal range search in $d \geqslant 3$ dimensions, Chazelle and Guibas [9] gave a solution in $O(s+\log d)$ query time using a data structure which requires $O(n \log^{d} n)$ space and can be constructed in $O(n \log^{d+1} n)$ time. Furthermore, for $d \geqslant 3$ dimensions a simple solution is given in [2], in $O(n \log^{d-1} n)$ preprocessing time and space and $O(s+\log^{d-2} n)$ query time. Bentley and Maurer propose in [4] three data structures for range searching in d dimensions. The first data structure has $O(n^{2d-1})$ preprocessing time and space, and $O(s+d \log n)$ query time. Also, they demonstrate that this query time is optimal under comparison-based models. The performance of the second data structure is $O(n^{1+\varepsilon})$ preprocessing time and space (for any fixed $\varepsilon > 0$), and $O(s+\log n)$ query time. The third data structure is constructed in $O(n \log n)$ time and requires $O(n)$ storage while the query time is $O(n^{\varepsilon})$ ($\varepsilon > 0$ can be chosen arbitrarily).

In Table I the computational complexity of all the above mentioned approaches is summarized.

Thus, the assignment of patterns to a d-range needs $O(s+\log^{d-2} n)$ time, where s is the number of patterns that lie within the d-range. Note that the

TABLE I

Methods for Orthogonal Range Search with the Corresponding Time and Space Complexity

Method	*Preprocessing time, space*	*Query time*
Range tree [20]	$O(n \log^{d-1} n), O(n \log^{d-1} n)$	$O(s+\log^{d} n)$
Wilard and Lueeker [20]	$O(n \log^{d-1} n), O(n \log^{d-1} n)$	$O(s+\log^{d-1} n)$
Multidimensional binary tree [20]	$\theta(dn \log n), \theta(dn)$	$O(s+dn^{-(1/d)})$
Chazelle [8]	$O(n \log^{d-1} n), O\left(n \dfrac{\log^{d-1} n}{\log \log n}\right)$	$O(s+\log^{d} n)$
Chazelle and Guibas [9]	$O(n \log^{d+1} n), O(n \log^{d} n)$	$O(s+\log^{d-2} n)$
Alevizos [2]	$O(n \log^{d-1} n), O(n \log^{d-1} n)$	$O(s+\log^{d-2} n)$
Bentley and Maurer [4]	$O(n^{2d-1}), O(n^{2d-1})$	$O(s+d \log n)$
	$O(n^{1+\varepsilon}), O(n^{1+\varepsilon})$	$O(s+\log n)$
	$O(n \log n), O(n)$	$O(n^{\varepsilon})$

area of d-ranges is small enough, so that $s \ll n$. Therefore, in each movement (iteration), the first loop of k-windows, where the patterns are assigned to d-ranges, has time complexity $O(k(s+\log^{d-2} n))$. The second loop, where the means of d-ranges are calculated, needs $O(sdk)$ time with arithmetic addition as the basic operation. Clearly, $sk \ll n$. Finally, the quality function is, also, calculated in $O(sdk)$ with arithmetic addition as the basic operation. Thus, the whole proposed algorithm has time complexity $O(dkqr(\log^{d-2} n/d+s))$ where q is the number of movements (iterations) and r is the number of repetitions caused by missing clusters. Notice that the basic operation is the arithmetic comparison between two numbers without any distance computation. Therefore, the k-windows algorithm has a significantly superior performance than the direct k-means algorithm.

It is worth mentioning that, although preprocess time and space complexity does not affect the performance of the algorithm, the data structure is constructed in $O(n \log^f n)$ time and space, where $d-1 \leqslant f \leqslant d+1$ depending on the chosen algorithm (see Table I).

5. EMPIRICAL RESULTS

In order to evaluate the proposed k-windows algorithm, we have implemented a system, in Borland C++Builder. Using this system, we have applied k-windows in three synthetic sample databases. The sample databases (DSet1, DSet2, DSet3) are depicted in Fig. 5 and they introduce

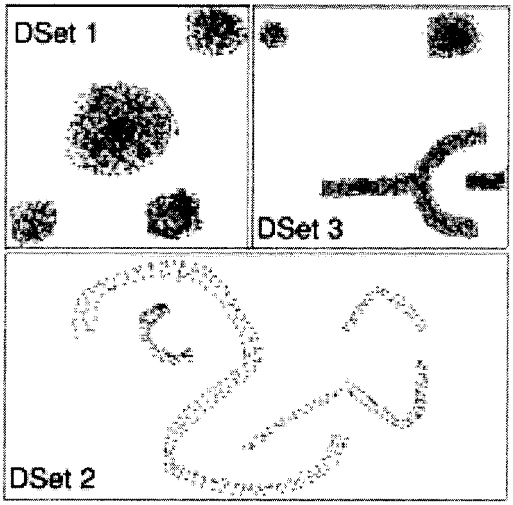

FIG. 5. The three synthetic sample databases DSet1, DSet2, and DSet3.

clusters with both normal and irregular shape. They have already been used as test data sets [25] to evaluate CLARANS, a clustering algorithm with a distance-based neighborhood definition, and DBSCAN, a density-based clustering algorithm. We have, also, applied *k*-means to these data sets, therefore we were able to compare *k*-windows with the three most popular clustering algorithms.

Empirical tests aim at examining the proportion between t and qr, so as to provide experimental evidence of the improvement achieved in the performance. Moreover, the improvement in partitioning accuracy is addressed from an experimental perspective using the three synthetic sample databases. Notice that, since the three clustering algorithms, that *k*-windows is compared with, are of different types, they have no common quantitative measure of the partition accuracy. Therefore, we evaluate their partitioning accuracy by visual inspection.

In Fig. 6 clusters discovered by *k*-means, CLARANS, DBSCAN and *k*-windows are shown for $k = 4$, in the first synthetic sample database. Clusters discovered by these algorithms in the other two synthetic sample databases are shown in Figs. 7 and 8, respectively. Clusters discovered by CLARANS, DBSCAN are taken from [25], where the evaluation methodology is described. Clusters discovered by *k*-means and *k*-windows are taken from our implementation of the algorithms. Note that, as far as the *k*-means algorithm is concerned, we used a primitive initialization method, that consists in preexecuting *k*-means in a sample of the data in order to

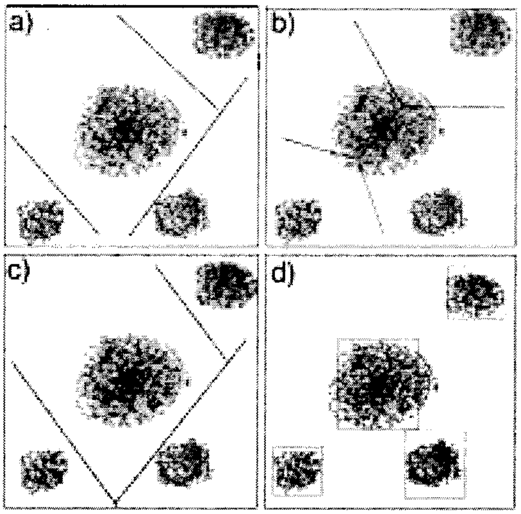

FIG. 6. Clusters discovered by (a)*k*-means, (b) CLARANS, (c) DBSCAN, and (d) *k*-windows in DSet1.

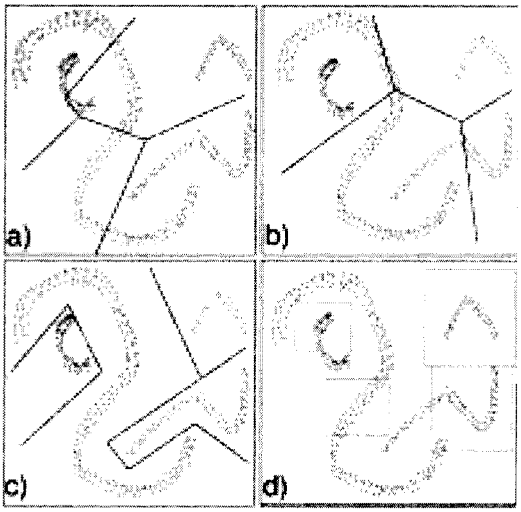

FIG. 7. Clusters discovered by (a)*k*-means, (b) CLARANS, (c) DBSCAN, and (d) *k*-windows in DSet2.

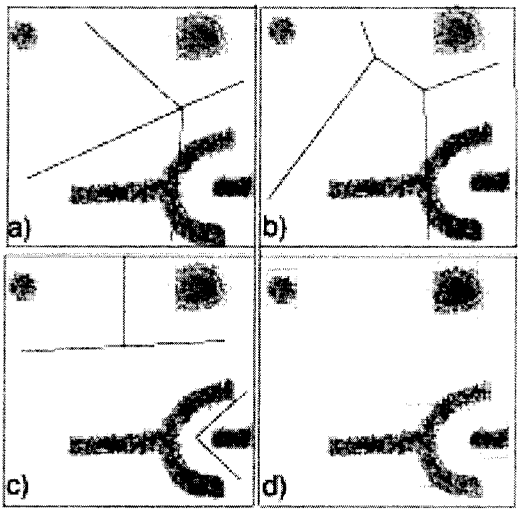

FIG. 8. Clusters discovered by (a)*k*-means, (b) CLARANS, (c) DBSCAN, and (d) *k*-windows in DSet3.

TABLE II

Processing Time along with n, t, q (Moves + Enlargements), r for k-Means, and k-Windows for $k = 4$

	Processing time	n	t	q	r
k-means in DSet1	0.14	1599	3		
k-windows in DSet1	0.03	1599		109 + 90	1
k-means in DSet2	0.01	409	4		
k-windows in DSet2	0.01	409		80 + 43	1
k-means in DSet3	0.16	1829	3		
k-windows in DSet3	0.03	1829		105 + 83	1

refine the initial points. Note, also, that the processing time of this step is not added to the processing time of the k-means algorithm shown in Tables II, III. Moreover, if this primitive initialization method were not used, partition accuracy of k-means algorithm would be worse. As far as the k-windows algorithm is concerned, we use the range tree in order to define the initial d-ranges.

In Table II the processing time along with n, t, q (moves + enlargements) and r is depicted for k-means and k-windows in the above data sets.

Clusters discovered by k-means and k-windows algorithms in the data sets for $k = 5$ are shown in Figs. 9, 10, and 11, respectively.

In Table III the processing time along with n, t, q (moves + enlargements), r is depicted for k-means and k-windows in the above data sets for $k = 5$.

Finally, in Fig. 12 the speedup, with respect to k-means, is depicted as a function of the size of the data set for $k = 4$ and $k = 5$. We can conclude that the speedup is linear to the size of the data set.

TABLE III

Processing Time along with n, t, q (Moves + Enlargements), r for k-Means, and k-Windows for $k = 5$

	Processing time	n	t	q	r
k-means in DSet1	0.15	1599	3		
k-windows in DSet1	0.1	1599		109 + 90	1
k-means in DSet2	0.01	409	3		
k-windows in DSet2	0.01	409		91 + 61	1
k-means in DSet3	0.17	1829	4		
k-windows in DSet3	0.07	1829		126 + 94	1

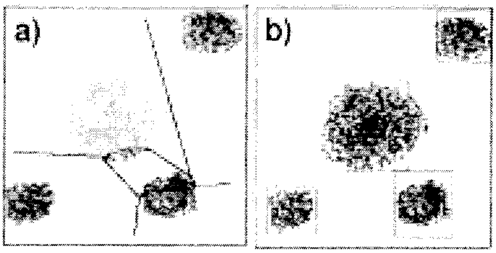

FIG. 9. Clusters discovered by (a) *k*-means and (b) *k*-windows in DSet1 for $k = 5$.

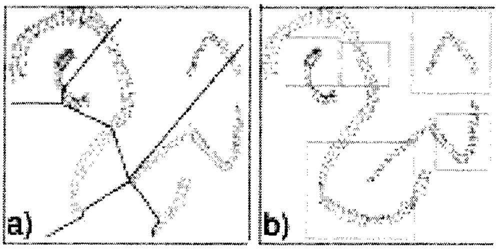

FIG. 10. Clusters discovered by (a) *k*-means and (b) *k*-windows in DSet2 for $k = 5$.

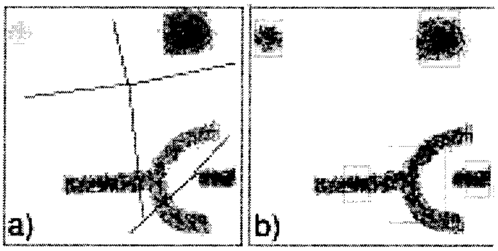

FIG. 11. Clusters discovered by (a) *k*-means and (b) *k*-windows in DSet3 for $k = 5$.

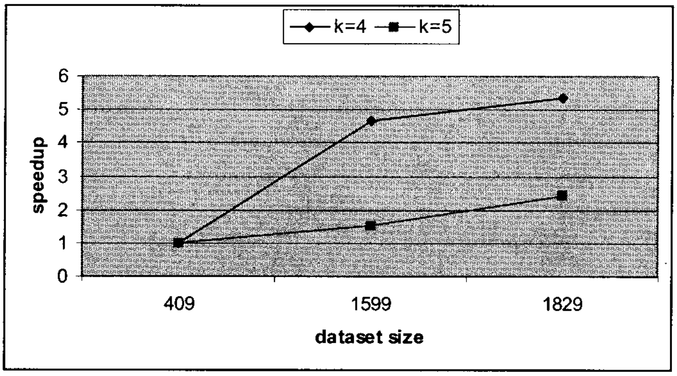

FIG. 12. Speedup as a function of the size of the data set.

6. CONCLUSIONS AND FUTURE WORK

The k-windows algorithm is an improvement of the well known k-means clustering algorithm, aiming at a better time complexity and partitioning accuracy. The time complexity of the k-means is $O(ndkt)$ while in our k-windows it is reduced to $O(dkqr(\log^{d-2} n/d+s))$. This is accomplished by reducing the number of patterns that need to be examined for similarity, in each iteration, using a windowing technique, that is based on range search. Moreover, our approach, in a meta-iteration phase, tries to further increase the partitioning accuracy.

It seems that the proposed k-windows algorithm would not be directly applicable in practical settings due to the superlinear space requirements for the range tree. However, one could follow several approaches for scaling up to very large data sets, as sampling (e.g., [6, 7]), or parallelizing (e.g., [16, 17], or distributing [5].

We are, currently, working on improving the performance of the meta-iteration phase using various approaches and on the design of more effective techniques for choosing initial windows. Moreover, we are working on extending the k-windows algorithm to work on categorical data. In addition, we are working on a parallel version of the proposed k-windows algorithm, assigning a different processor for each window. Note that, under such an assignment scheme, k-windows can efficiently be parallelized, in contrast to k-means that would require a large communication overhead. This is because, a processor dedicated to a cluster, in k-means, must be synchronized with all others before the assignment of a pattern. There is no such need in k-windows where the decision of assigning a pattern to a d-range is taken by each processor independently. However, since the d-ranges are enlarged in a parallel setting, there may be overlaps between them and thus merging problems have to be resolved.

We intend to address the above approaches in a future correspondence.

ACKNOWLEDGMENTS

We thank the anonymous referees for their constructive comments, suggestions, and invaluable criticisms which helped us to improve the paper. Moreover, we thank one of the referees for his strong suggestion to provide further results in future works. We also thank Mr. D. Tasoulis for his invaluable help during the tests.

REFERENCES

1. M. Aldenderfer and R. Blashfield, “Cluster Analysis,” Sage, Thousand Oaks, CA, 1984.
2. P. Alevizos, An algorithm for orthogonal range search in $d \geqslant 3$ dimensions, *in* “Proceedings of 14th European Workshop on Computational Geometry, Barcelona,” 1998.
3. K. Alsabti, S. Ranka, and V. Singh, An efficient k-means clustering algorithm, *in* “Proceedings of the First Workshop on High Performance Data Mining,” 1995.

4. J. L. Bentley and H. A. Maurer, Efficient worst-case data structures for range searching, *Acta Inform.* **13** (1980), 1551–1568.
5. B. Boutsinas and T. Gnardellis, On distributing the clustering process, *Pattern Recognition Letters* **23**, No. 8 (2002), 999–1008.
6. P. S. Bradley and U. M. Fayyad, Refining initial points for *k*-means clustering, *in* "Proceedings of the IJCAI-93, San Mateo, CA," pp. 1058–1063, 1983.
7. P. S. Bradley, U. M. Fayyad, and C. Reina, Scaling clustering algorithms to large databases, *in* "Proceedings of the 4th Int. Conf. on Knowledge Discovery and Data Mining," pp. 9–15, 1998.
8. B. Chazelle, Filtering search: A new approach to query-answering, *SIAM J. Comput.* **15**, No. 3 (1986), 703–724.
9. B. Chazelle and L. J. Guibas, Fractional cascading. II. Applications, *Algorithmica* **1** (1986), 163–191.
10. U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "Advances in Knowledge Discovery and Data Mining," MIT Press, Cambridge, MA, 1996.
11. J. C. Gower, A general coefficient of similarity and some of its properties, *BioMetrics* **27** (1971), 857–874.
12. M. J. Greenacre, "Theory and Applications of Correspondence Analysis," Academic Press, San Diego, 1984.
13. Z. Huang, Extensions to the *k*-means algorithm for clustering large data sets with categorical values, *Data Mining Knowledge Discovery* **2** (1998), 283–304.
14. A. K. Jain and R. C. Dubes, "Algorithms for Clustering Data," Prentice–Hall, Englewoods Cliffs, NJ, 1988.
15. D. Judd, P. McKinley, and A. Jain, Large-scale parallel data clustering, *in* "Proceedings of Int. Conference on Pattern Recognition," 1996.
16. X. Li and Z. Fang, Parallel clustering algorithms, *Parallel Comput.* **11** (1989), 275–290.
17. D. W. Piftzner, J. K. Salmon, and T. Sterling, Halo world: Tools for parallel cluster finding in astrophysical N-body simulations, *Data Mining Knowledge Discovery* **2**, No. 2 (1998), 419–438.
18. G. Piatetsky-Shapiro and W. Frawley, "Knowledge Discovery in Databases," AAAI Press, Menlo Park, CA, 1991.
19. C. Pizzuti, D. Talia, and G. Vonella, A divisive initialization method for clustering algorithms, *in* "Proc. PKDD'99—Third Europ. Conf. on Principles and Practice of Data Mining and Knowledge Discovery," Lecture Notes in Artificial Intelligence, Vol. 1704, pp. 484–491, Springer-Verlag, Prague, 1999.
20. F. Preparata and M. Shamos, "Computational Geometry," Springer-Verlag, New York/Berlin, 1985.
21. J. B. MacQueen, Some methods for classification and analysis of multivariate observations, *in* "Proceedings of the 5th Berkeley Symposium on Mathematics Statistics and Probability," pp. 281–297, 1967.
22. N. Ralambondrainy, A conceptual version of the *k*-means algorithm, *Pattern Recognition Lett.* **16** (1995), 1147–1157.
23. V. Ramasubramanian and K. Paliwa, Fast *k*-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding, *IEEE Trans. Signal Process.* **40**, No. 3 (1992).
24. E. H. Ruspini, A new approach to clustering, *Inform. and Control* **15** (1969), 22–32.
25. J. Sander, M. Ester, H. Kriegel, and X. Xu, Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications, *Data Mining Knowledge Discovery* **2** (1998), 169–194.
26. S. Z. Selim and M. A. Ismail, *k*-means-type algorithms: A generalized convergence theorem and characterization of local optimality, *IEEE Trans. Pattern Anal. Mach. Intelligence* **6**, No. 1 (1984), 81–87.

Printed in Belgium