5

Novel Approaches to Unsupervised Clustering Through *k*-Windows Algorithm

D.K. Tasoulis and M.N. Vrahatis*

Computational Intelligence Laboratory, Department of Mathematics,
University of Patras Artificial Intelligence Research Center (UPAIRC),
University of Patras, GR–26110 Patras, Greece.

Summary. The extraction of meaningful information from large collections of data is a fundamental issues in science. To this end, clustering algorithms are typically employed to identify groups (clusters) of similar objects. A critical issue for any clustering algorithm is the determination of the number of clusters present in a dataset. In this contribution we present a clustering algorithm that in addition to partitioning the data into clusters, it approximates the number of clusters during its execution. We further present modifications of this algorithm for different distributed environments, and dynamic databases. Finally, we present a modification of the algorithm that exploits the fractal dimension of the data to partition the dataset.

1 Introduction

Clustering is a fundamental field of explanatory data analysis that aims at discovering hidden structure in datasets. More specifically, clustering partitions a set of objects in groups (clusters) such that objects within the same group bear a closer similarity to each other, than objects in different groups. Clustering techniques have a very broad application domain including data mining [22], statistical data analysis [2], compression and vector quantization [40], global optimization [8, 54], web personalization [41] and text mining [19, 45].

The first comprehensive foundations of these methods was published in 1939 [55], but the earliest references date back to the fourth century B.C. by Aristotle and Theophrastos and in the 18th century to Linnaeus [30].

Following [1], to define more formally the clustering problem firstly, we assume that S is a set of n points in a d–dimensional metric space $(\mathbb{R}^d, \varrho)$. A *k-clustering* of S for an integer $k \leqslant n$ is defined as a partition Σ of S into k subsets $S_1, \ldots, S_k$, each one representing a different *cluster*. The *size of a cluster* S_i is defined as the maximum distance, under the ϱ metric, between a fixed point c_i, called the *center of the cluster*, and any other point of

* *Corresponding author:* M.N. Vrahatis, email: vrahatis@math.upatras.gr

D.K. Tasoulis and M.N. Vrahatis: *Novel Approaches to Unsupervised Clustering Through k-Windows Algorithm*, StudFuzz **185**, 51–77 (2005)
www.springerlink.com

S_i. Similarly, the *size of a k-clustering* Σ, is defined as the maximum cluster size among all the clusters in Σ [1]. The *k-center* problem is defined as the computation of a k-clustering of the smallest possible size. The k-center problem can also be formulated as covering S by congruent disks of the smallest possible size. Sometimes the centers of the clusters are required to be a subset of S. This requirement defines the *discrete k-center* problem. In some applications the number of points in each cluster is also important. Thus, if we define, for an integer $L > 0$, the *L-capacitized k-clustering* of S to be a partition Σ of S in k clusters with no cluster containing more than L points; then the *L-capacitized k-center* problem is defined as the computation of the L-capacitized k-clustering having the smallest possible size [1].

Clustering is a difficult scientific problem, since even the simplest clustering problems are known to be NP-Hard [1]. The Euclidean k-center problem in the plane is NP-Hard [33]. In fact, it is NP-Hard to approximate the two-dimensional k-center problem even under the L_∞-metric [23].

Irrespectively of the method used, a fundamental issue in cluster analysis is the determination of the number of clusters present in a dataset. This issue remains an open problem in cluster analysis. For instance well–known and widely used iterative techniques, such as the k-means algorithm [24], require from the user the a priori designation of the number of clusters present in the data.

To this end, we present the unsupervised k–windows clustering algorithm. This algorithm, by employing windowing techniques, attempts not only to discover the clusters but also their number, in a single execution. Assuming that the dataset lies in d dimensions, the algorithm initializes a number of d–dimensional windows (boxes), over the dataset. Subsequently, it iteratively moves and enlarges these windows in order to cover the existing clusters.

The approximation of the number of clusters is based on the idea of considering a large number of initial windows. The windowing technique of the k-windows algorithm allows for a large number of initial windows to be examined, without a significant overhead in time complexity. Once movement and enlargement of all windows terminate, all overlapping windows are considered for merging. The merge operation determines whether two windows belong to same cluster by examining the proportion of points in the overlapping area to the total number of points in each window. Thus, the algorithm is capable of providing an approximation to the actual number of clusters.

Database technology has enabled organizations to collect data at a constantly increasing rate. The development of algorithms that can extract knowledge in the form of clustering rules from such distributed databases has become a necessity. Distributed clustering algorithms attempt to merge computation with communication and explore all facets of the distributed clustering problem. The k-windows algorithm can be extended to a distributed environment.

Considering a non-stationary environment where update operations on the database are allowed, maintaining a cluster result at a low computational cost

becomes important. Utilizing a recently proposed dynamic data structure, we present an extension of the k-windows algorithm suitable for cluster maintenance.

An important property that describes the complexity of a dataset is its fractal dimension. Incorporating estimates of the fractal dimension in the workings of the k-windows algorithm, allows it to extract qualitative information for the underlying clusters.

The rest of the paper is organized as follows. The details of the unsupervised k-windows are described in Section 2. Next, in Section 3 two distributed versions of the algorithm are presented. Section 4 presents an extension of the algorithm to non-stationary environments. A modification of the algorithm that uses the fractal dimension is presented in Section 5. In Section 6 computational experiments are presented that demonstrate the applicability of the algorithm, on various datasets. The paper ends with concluding remarks in Section 7.

2 The unsupervised k-windows clustering algorithm

The k-windows clustering algorithm aims at capturing all the patterns that belong to one cluster within a d–dimensional window [56]. To this end, it employs two fundamental procedures: *movement* and *enlargement*. The movement procedure aims at positioning each window as close as possible to the center of a cluster. During this procedure each window is centered at the mean of the patterns that are included in it. The movement procedure is iteratively executed as long as the distance between the new and the previous center exceeds the user–defined variability threshold, θ_v. On the other hand, the enlargement process tries to augment the window to include as many patterns from the current cluster as possible. Thus, the range of each window, for each coordinate separately is enlarged by a proportion θ_e/l, where θ_e is user–defined and l stands for the number of previous valid enlargements. Valid enlargements are those that cause a proportional increase in the number of patterns included in the window, exceeding the user–defined coverage threshold, θ_c. Further, before each enlargement is examined for validity the movement procedure is invoked. If an enlargement for coordinate $c' \geqslant 2$, is considered valid, then all coordinates c'', such that $c'' < c'$, undergo enlargement assuming as initial position the current position of the window. Otherwise, the enlargement and movement steps are rejected and the position and size of the d–range are reverted to their prior to enlargement values. In Fig. 1 the two processes are illustrated.

As previously mentioned, a critical issue in cluster analysis, is the determination of the number of clusters that best describe a dataset. The unsupervised k-windows algorithm has the ability to provide an approximation to this number. The key idea is to initialize a large number of windows. After movement and enlargement of all windows terminates, all overlapping

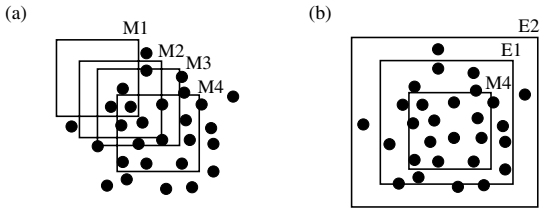

Fig. 1. (a) Sequential movements M2, M3, M4 of initial window M1. (b) Sequential enlargements E1, E2 of window M4.

windows are considered for merging. During this operation, for each pair of overlapping windows, the number of patterns that lie in their intersection is computed. Next, the proportion of this number to the total number of patterns included in each window is calculated. If this proportion exceeds a user defined threshold, θ_s, the two windows are considered to be identical and the one containing the smaller number of points is disregarded. Otherwise, if the mean exceeds a second user defined threshold, θ_m, the windows are considered to have captured portions of the same cluster and are merged. An example of this operation is exhibited in Fig. 2; the extent of overlapping of windows W1 and W2 exceeds the θ_s threshold, and W1 is deleted. On the other hand, windows W3 and W4 are considered both to belong to the same cluster. Finally, windows W5 and W6, are considered to capture two different clusters.

An example of the overall workings of the algorithm is presented in Fig. 3; In Fig. 3(a) a dataset that consists of three clusters is shown, along with six initial windows. In Fig. 3(b) after the merging operation the algorithm has correctly identified the three clusters.

The computationally demanding step of the k-windows clustering algorithm is the determination of the points that lie in a specific window. This is the well studied *orthogonal range search* problem [38]. Formally this problem can be defined as follows:

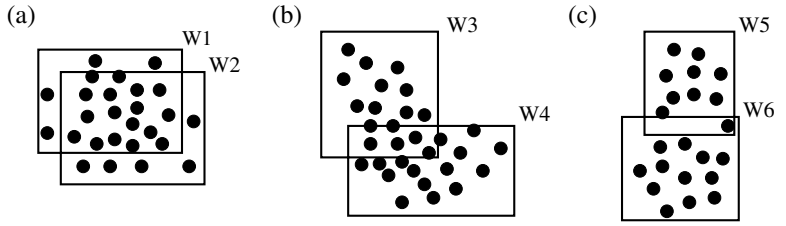

Fig. 2. (a) W1 and W2 satisfy the similarity condition and W1 is deleted. (b) W3 and W4 satisfy the merge operation and are considered to belong to the same cluster. (c) W5 and W6 have a small overlapment and capture two different clusters.

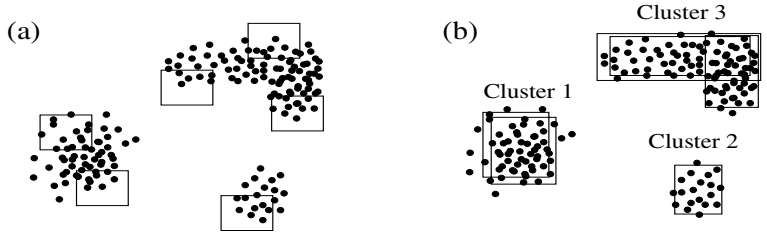

Fig. 3. An example of the application of the k-windows algorithm.

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

Numerous Computational Geometry techniques have been proposed to address this problem. All these techniques implicate a preprocessing stage at which they construct a data structure storing the patterns. This data structure allows them to answer range queries fast. In Table 1 the computational complexity of various such approaches is summarized. In detail, for applications of very high dimensionality, data structures like the Multidimensional Binary Tree [38], and Bentley and Maurer [10] seem more suitable. On the other hand, for low dimensional data with a large number of points the approach of Alevizos [3] appears more attractive.

	Preprocessing		
Method	**Time**	**Space**	**Query time**
Multidim. Binary Tree [38]	$\theta(dn \log n)$	$\theta(dn)$	$O\left(s + dn^{1-1/d}\right)$
Range Tree [38]	$O\left(n \log^{d-1} n\right)$	$O\left(n \log^{d-1} n\right)$	$O\left(s + \log^{d} n\right)$
Wilard and Lueeker [38]	$O\left(n \log^{d-1} n\right)$	$O\left(n \log^{d-1} n\right)$	$O\left(s + \log^{d-1} n\right)$
Chazelle [15]	$O\left(n \log^{d-1} n\right)$	$O\left(n \frac{\log^{d-1} n}{\log \log n}\right)$	$O\left(s + \log^{d-1} n\right)$
Chazelle and Guibas [16]	$O\left(n \log^{d+1} n\right)$	$O\left(n \log^{d} n\right)$	$O\left(s + \log^{d-2} n\right)$
Alevizos [3]	$O\left(n \log^{d-1} n\right)$	$O\left(n \log^{d-1} n\right)$	$O\left(s + \log^{d-2} n\right)$
Bentley and Maurer [10]	$O\left(n^{2d-1}\right)$	$O\left(n^{2d-1}\right)$	$O(s + d \log n)$
Bentley and Maurer [10]	$O\left(n^{1+\varepsilon}\right)$	$O\left(n^{1+\varepsilon}\right)$	$O(s + \log n)$
Bentley and Maurer [10]	$O(n \log n)$	$O(n)$	$O(n^{\varepsilon})$

Table 1. Methods for orthogonal range search with the corresponding time and space complexity (n is the number of points, d is their dimension and s is the result of the query).

Based on the above discussion we propose the following high level description of the algorithm:

Unsupervised k-windows clustering algorithm $(a, \theta_e, \theta_m, \theta_c, \theta_v, k)$ $\{$
  execute W=DetermineInitialWindows(k,a)
  for each d–range w_j in W **do**
    repeat
      execute movement(θ_v,w_j)
      execute enlargement(θ_e,θ_c,θ_v,w_j)
    until the center and size of w_j remain unchanged
  execute merging(θ_m,θ_s,W)
  Output $\{$clusters $c_{l1}, c_{l2}, \ldots$ such as: $c_{li} = \{i : i \in w_j, label(w_j) = li\}$ $\}$
$\}$
function DetermineInitialWindows(k,a) $\{$
  initialize k d–ranges $w_{m1}, \ldots, w_{mk}$ each of size a
  select k random points from the dataset and
    center the d-ranges at these points
  return a set W of the k d–ranges
$\}$
function movement(θ_v, a d–range w) $\{$
  repeat
    find the patterns that lie within the d–range w
    calculate the mean m of these patterns
    set the center of w equal to m
  until the distance between m and the previous center of w is less than θ_v
$\}$
function enlargement(θ_e,θ_c,θ_v,a d–range w) $\{$
  repeat
    foreach coordinate d_i **do**
      repeat
        enlarge w across d_i for θ_e%
        execute movement(θ_v,w)
      until increase in number of patterns across d_i is less than θ_c%
  until increase in number of patterns is less than θ_c% across every d_i
$\}$
function merging(θ_m,θ_s,a set W of d–ranges) $\{$
  for each d–range w_j in W not marked **do**
  mark w_j with label w_j
  if $\exists\ w_i \neq w_j$ in W, that overlaps with w_j
    compute the number of points n that lie in the window overlapment
    if $n/|w_i| \geqslant \theta_s$ and $|w_i| < |w_j|$
      disregard w_i
    if $0.5\,(n/|w_j| + n/|w_i|) \geqslant \theta_m$
      mark all w_i labeled d–ranges in W with label w_j
$\}$

3 Distributing the clustering process

The ability to collect, store and retrieve data has been constantly increasing throughout the past decades. This fact has rendered the development of algorithms that can extract knowledge in the form of clustering rules from various databases simultaneously, a necessity. This trend has been embraced by distributed clustering algorithms, that attempt to merge computation with communication and explore all facets of the distributed clustering problems.

Although several approaches have been introduced for parallel and distributed Data Mining [13, 25, 28], parallel and distributed clustering algorithms have not been extensively studied. In [58] a parallel version of DBSCAN [43] and in [18] a parallel version of k-means [24] were introduced. Both algorithms start with the complete data set residing in one central server and then distribute the data among the different clients. For instance, in the case of parallel DBSCAN, data is organized at the server site within an R*-tree [9]. The preprocessed data are then distributed among the clients, which communicate with each other via messages.

Typically, in a distributed computing environment the dataset is spread over a number of different sites. Thus, let us assume that the entire dataset X is distributed among m sites each one storing X_i for $i = 1, \ldots, m$, so:

$$X = \bigcup_{i=1,\ldots,m} X_i.$$

Furthermore let us assume that there is a central site, C, that will hold the final clustering results.

At this point, different assumptions can be considered for the nature of communication among the sites. Primarily we can consider that the sites are connected through a high speed network, and data disclosure is allowed. On the other hand, a different assumption would enforce minimal communication among the sites. This could be due to privacy issues, or very slow and expensive network connections.

In the following paragraphs two version of the k-windows algorithm will be presented for the two opposing assumptions. Each version takes under consideration the underlying restrictions of the environment, and tries to provide efficient and effective clustering results.

3.1 Distributed clustering for minimal communication environments

Assuming an environment that enforces minimal communication, it is possible to modify the k-windows algorithm to distribute locally the whole clustering

procedure. In more detail, at each site i, the k-windows algorithm is executed over the X_i dataset. This step results in a set of d–ranges (windows) W_i for each site. To obtain the final clustering result over the whole dataset X, all the final windows from each site are collected to the central node C. The central node is responsible for the final merging of the windows and the construction of the final results. As it has already been mentioned in Section 2, all overlapping windows are considered for merging. The merge operation is based on the number of patterns that lie in the intersection of the windows.

This version of the algorithm assumes that the determination of the number of patterns at each intersection between two windows may be impossible. For example each site might not want to disclose this kind of information about its data. Alternatively, the exchange of data might be over a very slow network that restricts the continuous exchange of information. Under this constraint, the proposed implementation always considers two overlapping windows to belong to the same cluster, irrespective of the number of overlapping points. The θ_m and θ_s parameters become irrelevant. A high level description of the proposed algorithmic scheme follows:

Minimal communication distributed k-windows
for each site i, with $i = 1, \ldots, m$
execute the k-windows algorithm over X_i
send W_i to the central node C.
At the central node C
    for each site i
        get the resulting set of d–ranges W_i
        set $W \leftarrow W \cup W_i$
    {**comment**: d–range merging}
    for each d–range w_j not marked **do**
        mark w_j with label w_j
        if $\exists\, w_i \neq w_j$, that overlaps with w_j
        then mark w_i with label w_j

3.2 Distributed clustering over a fast communication network

Assuming that the sites involved in the distributing environment are connected through a fast network infrastructure, the algorithm can be modified to distribute the computational cost without imposing any restriction to its efficiency. More specifically, it is possible to distribute the computational effort of the k-windows algorithm by only parallelizing the range queries. In detail, assume again m computer nodes are available, each one having a portion of the dataset V_i where $i = 1, \ldots, m$. Firstly at each node, i, a multidimensional binary tree [38] T_i is constructed, which stores the points of the set V_i. Then parallel search for a range query Q is performed as follows:

Parallel range search procedure
set $A \longleftarrow \emptyset$
for each node i **do in parallel**
set $A_i \longleftarrow \emptyset$
find the points from the local database that are included in Q
insert the recovered points in A_i
send A_i to the server node
set $A \longleftarrow A \cup \{A_1, \ldots, A_m\}$

The algorithm at a preprocessing step constructs a multidimensional binary tree for each node holding data known only to that node. Then a server node is used to execute the k-windows algorithm. From that point onwards, the algorithm continues to work as in the original version. When a range search is to be executed, the server executes the range query over all the nodes and computes the union of the results.

To analyze the algorithms complexity, we assume that the multidimensional binary tree, is used as a data structure [38]. Then, the algorithmic complexity for the preprocessing step for n points in d dimensions is reduced to $\theta((dn \log n)/m)$ from $\theta(dn \log n)$ of the single node version. Furthermore the storage requirements at each node come up to $\theta(dn/m)$ while for the single node they remain $\theta(dn)$. Since the orthogonal range search algorithm has a complexity of $O(d\, n^{1-1/d} + s)$ [38], the parallel orthogonal range search algorithm has a complexity of $O(d\,(n/m)^{1-1/d} + s + \epsilon(d, m))$, where s is the total number of points included in the range search and $\epsilon(d, m)$ is a function that represents the time required for the communication between the master and the nodes. It should be noted that the only information that needs to be transmitted from each slave is the number of points found and their mean value as a d-dimensional vector. So the total communication comes to a broadcast message from the server about the range, and m messages of an integer and a d-dimensional vector from each slave. Taking these parameters under consideration, $\epsilon(d, m)$ can be computed for a specific network interface and a specified number of nodes. For the parallel algorithm to achieve an execution time speedup the following relation must hold:

$$O\left(\frac{d\left(\frac{n}{m}\right)^{1-\frac{1}{d}} + s + \epsilon(d, m)}{d\, n^{1-\frac{1}{d}} + s}\right) \leqslant 1,$$

which comes to [46]:

$$O(\epsilon(d, m)) \leqslant O\left(d\left(n^{1-\frac{1}{d}} - \left(\frac{n}{m}\right)^{1-\frac{1}{d}}\right)\right).$$

As long the above inequality holds, the parallel version of the algorithm is faster than the single node version. In all other cases the network infrastructure presents a bottleneck to the system. In that case, the parallel version advantage is limited to storage space requirements.

4 Clustering on dynamic databases

Most clustering algorithms rely on the assumption that the input data constitute a random sample drawn from a stationary distribution. As data is collected over time the underlying process that generates them can change. In a non–stationary environment new data are inserted and existing data are deleted. Cluster maintenance deals with the issues of when to to update the clustering result and how to achieve this with low computational cost. In the literature there are few maintenance algorithms, most of which are developed for growing databases. The application domain of these algorithms includes database re–organization [59], web usage user profiling [34] as well as, document clustering [57].

From the broader field of data mining, a technique for maintaining association rules in databases that undergo insertions and deletions has been developed in [17]. A generalization algorithm for incremental summarization has been proposed in [21]. An incremental document clustering algorithm that attempts to maintain clusters of small diameter as new points are inserted in the database has been proposed in [14]. Another on-line star-algorithm for document clustering has been proposed in [6]. A desirable feature of the latter algorithm is that it imposes no constraints on the number of clusters. An incremental extension to the GDBSCAN algorithm [43] has been proposed in [20]. Using a similar technique an incremental version of the OPTICS algorithm [5] has been proposed in [27]. The speedup achieved by this incremental algorithm [27] is significantly lower than that of [20]. This is attributed to the higher complexity of OPTICS, but [27] claim that the incremental version of OPTICS is suitable for a broader range of applications.

In the following paragraphs we present an extension of the unsupervised k-windows clustering algorithm [51, 53] that can efficiently mine clustering rules from databases that undergo insertion and deletion operations over time. The proposed extension incorporates the Bkd-tree structure [39]. The Bkd-tree can efficiently index objects under a significant load of updates, and also provides a mechanism that determines the timing of the updates.

4.1 The Bkd-tree

Considering databases that undergo a significant load of updates, the problem of indexing the data arises. In detail, an efficient index should be characterized by the properties of high space utilization and small processing time of queries under a continuous updating process. Moreover, the processing time of the updates must be fast. To this end, we employ the Bkd-tree structure proposed in [39], that maintains its high space utilization and excellent query and update performance, regardless of the number of updates performed.

The Bkd-tree is based on a well-known extension of the kd-tree (called the K-D-B-tree [42]) and on the so-called logarithmic method for making a static structure dynamic. Extensive experimental studies [39] have shown that

the Bkd-tree is able to achieve almost 100% space utilization and also the fast query processing of a static K-D-B-tree. However, unlike the K-D-B-tree, these properties are maintained under a massive load of updates.

Instead of maintaining one tree and dynamically re-balancing it after each insertion, the Bkd-tree structure maintains a set of $\log_2(n/M)$ static K-D-B-trees and updates are performed by rebuilding a carefully chosen set of structures at regular intervals (M stands for the capacity of the memory buffer, in terms of number of points). To answer a range query using the Bkd-tree, all the $\log_2(n/M)$ trees have to be queried. Despite this fact, the worst–case behavior of the query time is still of the order $O(d\, n^{1-1/d} + s)$ (s is the number of retrieved points). Using an optimal $O(n \log_m(n)$ bulk loading algorithm an insertion is performed in $O(\log_m(n) \log_2(n/M))$. A deletion operation is executed by simply querying each of the trees to find the tree T_i containing the point and delete it from T_i. Since there are at most $\log_2(n/M)$ trees, the number of operations performed by a deletion is $\log(n) \log_2(n/M)$ [39].

Insertions are handled completely differently. Most insertions ($(M-1)$ out of M consecutive ones) take place on the T_0 tree structure. Whenever T_0 reaches the maximum number of points it can store (M points) the smallest j is found such that T_j is an empty kd-tree. Then all points from T_0 and T_i for $0 \leqslant i < j$ are extracted and bulk loaded in the T_j structure. In other words, points are inserted in the T_0 structure and periodically reorganized towards larger kd-trees by merging small kd-trees into one large kd-tree. The larger the kd-tree, the less frequently it needs to be reorganized.

Extensive experimentation [39] has shown that the range query performance of the Bkd-tree is on par with that of existing data structures. Thus, without sacrificing range query performance, the Bkd-tree makes significant improvements in insertion performance and space utilization; insertions are up to 100 times faster than K-D-B-tree insertions and space utilization is close to a perfect 100%, even under a massive load of insertions.

4.2 Unsupervised *k*-windows on dynamic databases

The proposed dynamic version of the unsupervised k–windows algorithm is based on the utilization of the Bkd–tree data organization structure. The Bkd–tree primarily enables the fast processing of range queries, and secondly provides a criterion for the timing of the update operations on the clustering result. The following schema outlines the dynamic algorithm:

(a) Assume an execution of the algorithm on the initial database has been performed yielding a set of windows that describe the clustering result.
(b) At specified periods execute the following steps:
  (1) Treatment of insertion operations.
  (2) Treatment of deletion operations.
(c) After each of the above steps is completed update the set of windows.

This schema describes a dynamic algorithm that is able to adapt a clustering model to the changes in the database. In the following paragraphs after analyzing the workings of the algorithms for each possible update operation, a total high level description of the overall procedure is presented.

Treatment of insertions:

Insertion operations is the first update operation in a dynamic environment. Throughout this paragraph it is assumed that the static unsupervised k-windows algorithm has been applied on the initial database, producing a set of windows that describe the clustering result.

As insertion operations take place, the T_0 structure of the Bkd–tree reaches the maximum number of points it can store (M). At this point a number of windows are initialized over these points. Subsequently, the movement and enlargement procedures of the unsupervised k–windows algorithm are applied on these windows just as in the static case. When the movement and enlargement of the new windows terminate they are considered for similarity and merging with all the existing windows. Thus the algorithm is able to retain only the most representative windows, thereby restraining the clustering result to a relatively small size. An example of this procedure is demonstrated in Fig. 4. The filled circles represent the initial points while the empty circles represent the inserted points. The W1 and W2 windows are assumed to have been finalized from the initial run of the algorithm. On the other hand windows W3 and W4 are initialized over the inserted points (empty circles). After the completion of movement and enlargement operations for W3 and W4, they are considered for similarity and merging. This step yields that windows W1 and W3 belong to the same cluster, since they satisfy the merge operation, while, window W2 is ignored as it satisfies the similarity operation with window W4.

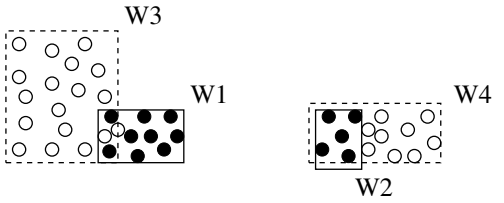

Fig. 4. The application of the k-windows algorithm over the inserted points.

Treatment of deletions:

The deletion update operations are addressed by maintaining a second Bkd-tree structure. Each time a point is deleted, it is removed from the main data

structure and is inserted in the second Bkd-tree. A number of windows are initialized over the points of the second Bkd-tree, when it reaches its maximum size. These windows are subjected to the movement and enlargement procedures of the k-windows algorithm, that operates only on the second data structure. When these operations terminate, the windows are considered for similarity with the windows that have been already processed. If a processed window is found to be similar with a window that contains deleted points, the former window is ignored. If the processed window that is ignored contained a large number of points new windows are initialized over these points and they are processed as new windows. After this procedure terminates, the Bkd-tree that stores the deleted points is emptied. An example of the deletion process is illustrated in Fig. 5. The filled circles represent the points that remain in the database, while the empty circles represent the deleted points. Window W1 is assumed to have been finalized from the initial run of the algorithm. Windows W2 and W3 are initialized over the deleted points (empty circles). After movement and enlargement, they are considered for similarity with the initial window, W1. Window W1 satisfies the similarity condition with W2 and W3 and thus it is ignored. Since window W1 contained a large number of points four windows are initialized over these points (Fig. 5(b)). The movement and enlargement operations on these yield windows W4, W5, W6 and W7. These windows are considered for merging and similarity. Windows W4 and W7 satisfy the similarity operation and thus window W7 is ignored. Windows W5 and W6 satisfy the merge operation thus they are considered to enclose points belonging to the same cluster.

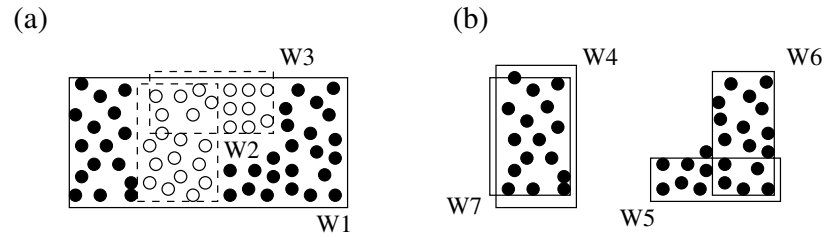

Fig. 5. (a) The application of the k-windows algorithm over the deleted points. (b) The application of the k-windows algorithm over the non-deleted points contained in initial window W1.

Performing the clustering operation on the deleted points the dynamic algorithm aims at identifying the windows that need to be re-organized in the initial results. Thus, the speedup that can be achieved depends not only on the size of the updates, but also on the change they impose on the clustering result.

Proposed algorithm:

Based on the procedures previously described, we propose the following high level algorithmic scheme:

Dynamic unsupervised k-windows
Set {the input parameters of k-windows algorithm}.
Initialize an empty set W of d–ranges.
Each time the T_0 tree of the Bkd-tree structure is full:
  Initialize a set I of k d–ranges, over the T_0 tree.
  Perform movements and enlargements of the d–ranges in I.
  Update W to contain the resulting d–ranges.
  Perform merging and similarity operations of the d–ranges in W.
If a large enough number of deletions has been performed:
  Initialize a set D of k d–ranges over the deleted points.
  Apply k-windows on the d–ranges in D.
  If any windows in D satisfy the similarity condition
  with windows in W:
    Then delete those windows from W and
    If the deleted windows from W contained any not deleted points,
    apply k-windows over them.
Report the groups of d–ranges that comprise the final clusters.

The above schema's execution is crucially affected by the size, M, of the T_0 component of Bkd-tree structure. The value of this parameter determines the timing of the update operation of the database [39], which in turn triggers the update of the clustering result. Therefore its value must be set according to the available computational power, the desired update intervals of the clustering result, as well as, the size of the application at hand.

5 Unsupervised clustering using fractal dimension

It is evident by a plain examination of the objects that surround us that most of them are very complex and erratic in nature [36, 44]. Mandelbrot [32], by introducing the concept of "fractal", was the first to try to address the need for a model that has the ability to describe such erratic behavior. A set is called fractal if its Hausdorff-Besicovitch dimension is strictly greater than its topological dimension. A characteristic for a fractal set is its *fractal dimension*, that measures its complexity. The box counting method [29], is an established approach to compute the fractal dimension of a set. In detail, for a set of n points in $\mathbb{R}^d$, and a partition of the space in grid cells of length l_b, the fractal dimension D_b is given by:

$$D_b = -\lim_{l_b \to 0} \frac{\log_{10} n_b(l_b)}{\log_{10} l_b},$$

where $n_b(l_b)$ represents the number of cells occupied by at least one point. D_b corresponds to the slope of the plot $\log_{10} n_b(l_b)$ versus $\log_{10} l_b$.

The fractal dimension has been utilized for clustering purposes in the past. A grid based clustering algorithm, that uses fractal dimension to cluster datasets, has been proposed by Barbarä and Chen [7]. The algorithm, uses a heuristic based algorithm at the initialization stage to form the initial clusters and then it incrementally adds points to a cluster, as long as, the fractal dimension remains constant. Another approach for two dimensions has been proposed by Prasad *et al.* [37]. Both algorithms require from the user to provide an *a priori* estimation of the number of clusters present in the dataset.

Next, we present a modification of the unsupervised k-windows clustering algorithm, that guides the procedures of movement, enlargement and merging using the fractal dimension of the points included in the window [52].

In detail, the movement and enlargement of a window is considered valid only if the associated change of the fractal dimension is not significant. It is also possible to guide the merging procedure by using the fractal dimension by allowing two windows to merge only if the estimated fractal dimensions are almost equal. Thus, the merging of windows that capture regions of a cluster with different fractal dimension is discouraged. Such clusters appear in datasets where the density of points in the neighborhood of the cluster center is significantly higher than that of areas located further away from the center. Thus, the algorithm discovers the cluster center more efficiently and moreover it identifies regions with qualitative differences within a single cluster. Consider for example the case exhibited in Fig. 6. The enlargement and movement procedures restrain window W3 from enclosing the right part of the cluster since the fractal dimension of this region is much higher (see Fig. 6(b)). Similarly, window W4 is restrained from capturing the left part of the cluster. The proposed modification of the algorithm also recognizes that although the windows have many points in common (see Fig. 6(a)), the difference in the value of the fractal dimension between them is sufficiently large so as to consider them as two distinct regions of the same cluster.

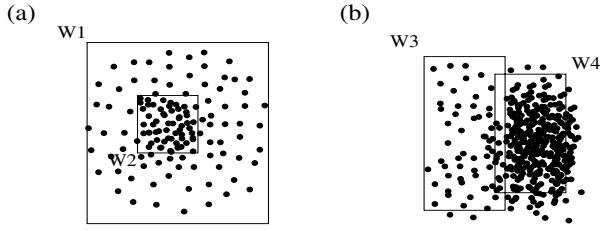

Fig. 6. Clusters with regions of different density. The proposed algorithm is able to discover the different sections of the same clusters.

6 Presentation of experiments

To evaluate the results of the unsupervised k-windows clustering algorithm we employ artificial datasets as well as real world ones. The first two datasets $Dset_1$ and $Dset_2$ are 2 dimensional, containing 1600 and 10000 points respectively. In $Dset_1$ the points are organized in 4 different clusters, of different sizes. On the other hand, $Dset_2$ contains 100 clusters of the same size (1000 points each). The centers of each cluster for this dataset are aligned over a grid in $[10, 200]^2$, and the corresponding points are drawn from a normal distribution with standard deviation along each dimension a random number between 1 and 2. The algorithm was applied on these two datasets with 12 and 256 initial windows respectively. The values of the parameters $\{\theta_e, \theta_m, \theta_c, \theta_v\}$ were set to $\{0.8, 0.1, 0.2, 0.02\}$ in both cases. k-windows was able to identify all the clusters correctly in both datasets. The datasets, as well as, the results obtained are illustrated in Fig. 7.

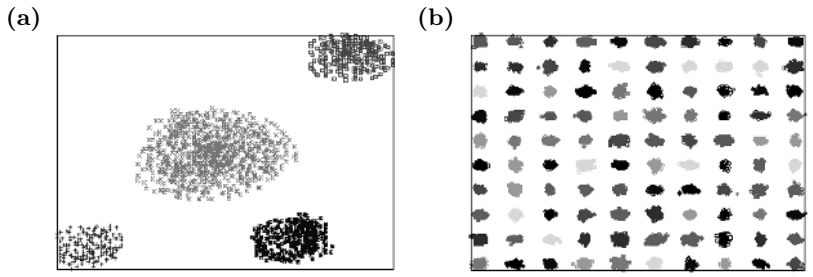

Fig. 7. (a) Results of the k-windows algorithm for $Dset_1$. (b) Results of the k-windows algorithm for $Dset_2$.

The next dataset $Dset_3$ is 3–dimensional, and is generated by uniformly sampling 20 cluster centers in the $[10, 200]^3$ range. Around each cluster center 100 points are sampled from a normal distribution with standard deviation along each dimension a random number between 1 and 3. $DSet_4$ is generated in a similar manner, but it lies in 50 dimensions. In both cases the algorithm initialized 128 windows while all other parameters were assigned to the same values as in the first two cases. In both cases the algorithm correctly identified the 20 clusters. This result is illustrated in Fig. 8.

The final two artificial datasets $Dset_5$ and $Dset_6$ consist of 319 and 3651 points respectively. Both of them contain 4 non-convex irregularly shaped clusters with uniformly scattered points. The application of the k-windows algorithm on them with the same initial values and 32 and 256 initial windows, is exhibited in Fig. 9. From this figure it is obvious that the algorithm is also able to discover clusters of irregular shapes as long as enough windows are initialized over the datasets.

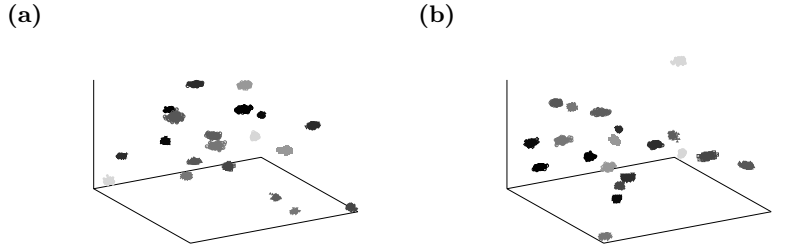

Fig. 8. (a) The result of the k-windows algorithm for $Dset_3$. (b) Results of the k-windows algorithm on a 3-dimensional projection of $Dset_4$

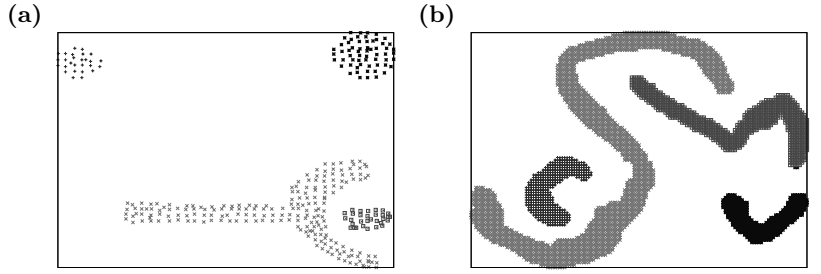

Fig. 9. (a) Results of the k-windows algorithm for $Dset_5$. (b) Results of the k-windows algorithm for $Dset_6$.

The real world dataset $Dset_7$ was part of the KDD 1999 Cup data set [26]. This dataset was generated by the 1998 DARPA Intrusion Detection Evaluation Program that was prepared and managed by MIT Lincoln Labs. The objective was to survey and evaluate research in intrusion detection. A standard set of data to be audited was provided, which includes a wide variety of intrusions simulated in a military network environment. The 1999 KDD intrusion detection contest uses a version of this dataset. For the purposes of this paper the 100000 first records of KDD 1999 Cup train dataset were used. This part of the data set contains 77888 of patterns of normal connections and 22112 of denial of service (DoS) attacks. Over the 42 features the 37 numeric ones were selected. When the algorithm is applied over this dataset with 16 initial windows it results in seven clusters out of which one contains 22087 DoS patterns. The other six clusters contain normal patterns exclusively, with the exception of one cluster that also contains 24 DoS patterns. These results point out that the discovered clusters are meaningful, and thus the clustering result can be considered accurate.

To test the efficiency of the distributed k-windows clustering algorithm for minimal communication environments described in Subsection 3.1, we resort to experiments that produce results that can be readily visualized. Thus, a two

dimensional dataset $Dset_d$ consisting of 10241 points was constructed. This dataset contains 5 clusters of different sizes, and a number of outlier points some of which connect two clusters. At a next step the dataset was randomly permuted and was distributed over 4, 8 and 16 sites. The k-windows clustering algorithm was applied to this dataset with 256, 64, 32 and 16 initial windows for the 1, 4, 8 and 16 sites respectively. The dataset, along with the clustering result, when the whole dataset resides in one single site and is distributed in 4 sites, is illustrated in Fig. 10. In Fig. 11 the results of the algorithm for 8 and 16 sites, respectively, are exhibited. As it is obvious from the figures that the results are correct in all three cases. It should be noted that for the cases of 8 and 16 sites a different extra cluster is identified by the algorithm, but it is not considered important since in both cases it holds a small amount of points and does not affect the correct identification of the 5 main clusters.

The next experimental results involve the measurement of the speedup that can be achieved through the distributed k-windows algoritm over a fast communication network, described in Subsection 3.2. To this end, we em-

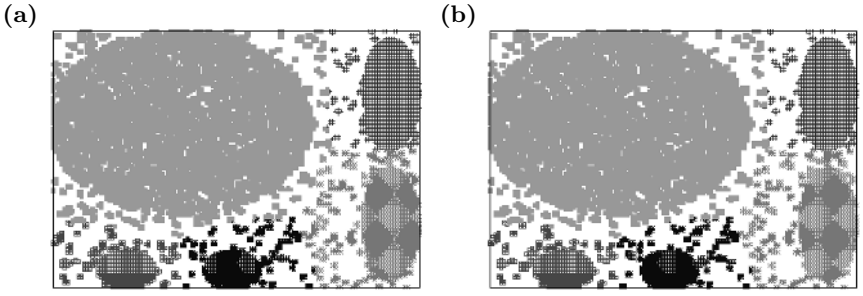

Fig. 10. (a) Results of the k-windows algorithm for $Dset_d$. (b) Results of the k-windows algorithm for $Dset_d$ for 4 sites and 64 initial windows per site.

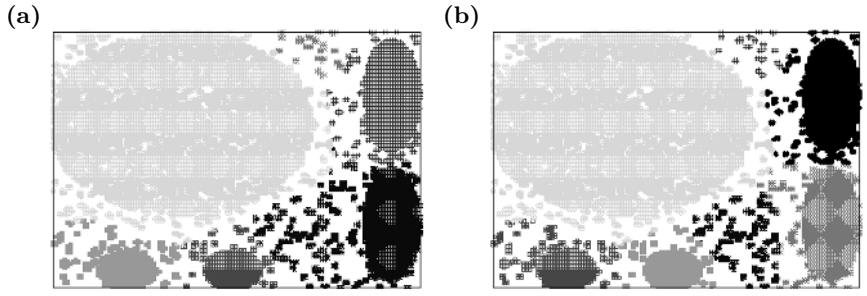

Fig. 11. (a) Results of the k-windows algorithm for $Dset_d$ for 8 sites and 32 initial windows per site. (b) Results of the k-windows algorithm for $Dset_d$ for 16 sites and 16 initial windows per site.

ploy the PVM parallel programming interface. PVM was selected, among its competitors because any algorithmic implementation is quite simple, since it does not require any special knowledge apart from the usage of functions and setting up the PVM process to all personal computers. Thus, the k-windows clustering algorithm was developed under the Linux operating system using the C++ programming language and its PVM extensions.

The hardware used for our purposes consisted of 16 Pentium III 900MHz personal computers with 32MB of RAM and 4GB of hard disk availability each. A Pentium 4 1.8GHz personal computer with 256MB of RAM and 20GB of hard disk availability was used as the server for the algorithm. The nodes were connected through a Fast Ethernet 100MBit/s network switch.

Furthermore, we constructed an artificial dataset $Dset_p$ using a mixture of Gaussian random distributions. The dataset contained 21000 points with 50 numerical attributes. As it is exhibited in Fig. 12, for this dataset the algorithm achieves almost 9 times smaller running time when using 16 CPUs. On the other hand, at every node only the 1/16 of the total storage space is required. From Fig. 12, we also observe an abrupt slow–down in speedup when moving from 8 to 16 nodes. This behavior is due to the larger number of messages that must be exchanged during the operation of the algorithm, which results to increased network utilization.

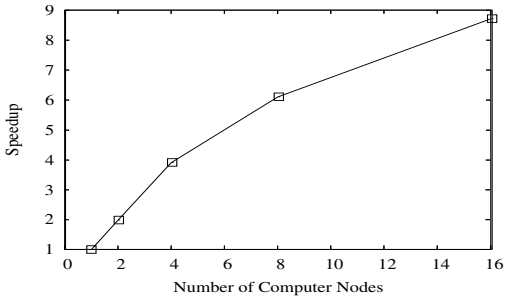

Fig. 12. Speedup for the different number of CPUs.

To evaluate the performance of the proposed dynamic k-windows algorithm we will use $Dset_1$. As previously mentioned, this dataset contains 1600 points organized in four clusters of different sizes. The dataset was split into four parts each part containing 400 points. The parts of the dataset were gradually presented to the algorithm. Each time a part was presented, the algorithm initialized a set of 32 windows over the new points. These windows were processed through the algorithmic procedure described above, and the clustering results for each step are presented in Fig. 13. In detail, in Fig. 13(a) and Fig. 13(b), seven clusters are identified, an outcome that appears to be reasonable by means of visual inspection. In Fig. 13(c) the algorithm detects

five clusters by correctly identifying the top right and bottom left clusters. The bottom right cluster is still divided into two clusters. Finally, in Fig. 13(d) all the clusters are correctly identified, by seven windows.

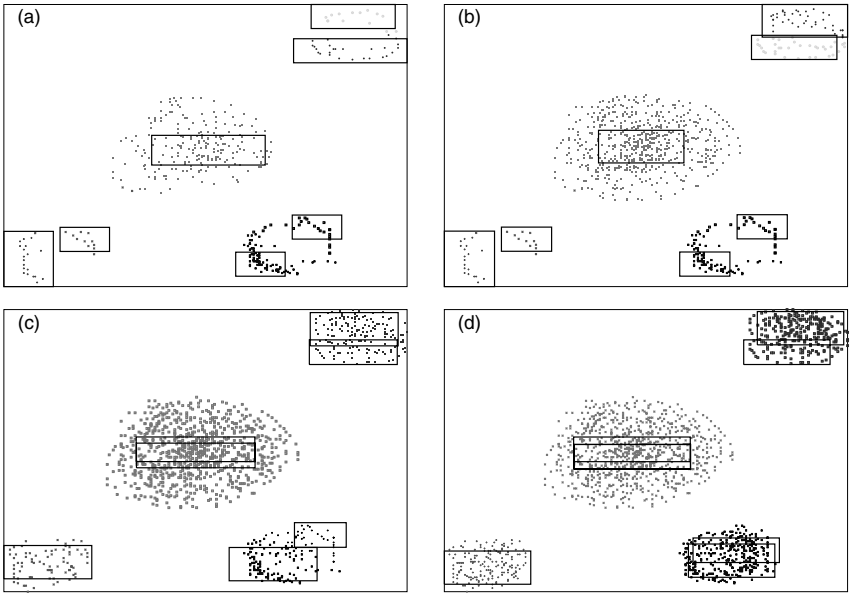

Fig. 13. Applying k-windows into four consecutive instances of $Dset_1$.

For comparative purposes with the work of [27, 43], we also calculated the speedup achieved by the dynamic version of the algorithm. To this end, we constructed a 10–dimensional dataset, $Dset_{online}$, by uniformly sampling 100 cluster centers in the $[10, 200]^{10}$ range. Around each cluster center 1000 points were sampled from a normal distribution with standard deviation along each dimension a random number in the interval $[1, 3]$. To measure the speedup we computed the CPU time that the static algorithm requires when it is re-executed over the updated database with respect to the CPU time consumed by the dynamic version. The results are exhibited in Figs. 14, and 15. For the insertions case (Fig. 14) the dynamic version manages to achieve a speedup factor of 906.96 when 100 insertion operations occur in database of original size 90000. For a larger number of insertion operations 1000 the speedup obtained although smaller 92.414 appears to be analogous to the ratio of the number of updates to the total size of the database.

For the case of deletions (Fig. 15) the speedup factors obtained are larger. For example when the size of the database is 900100 the speedup reaches 2445.23 and 148.934 for 100 and 1000 random deletions, respectively. It is important to note that in the case of deletions the speedup does not increase

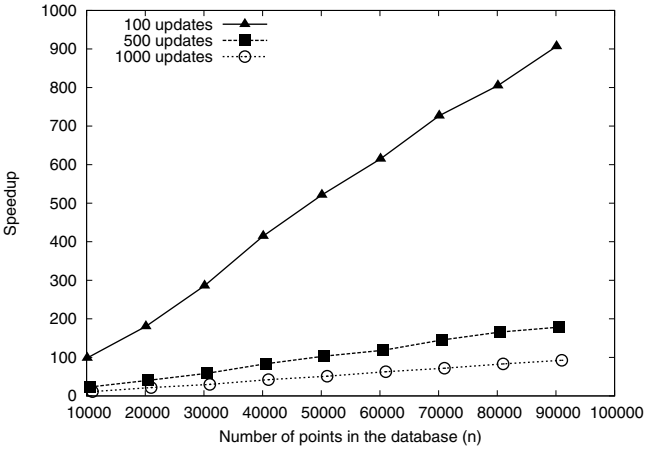

Fig. 14. Speedup achieved by the dynamic algorithm for insertion operations for $Dset_{online}$.

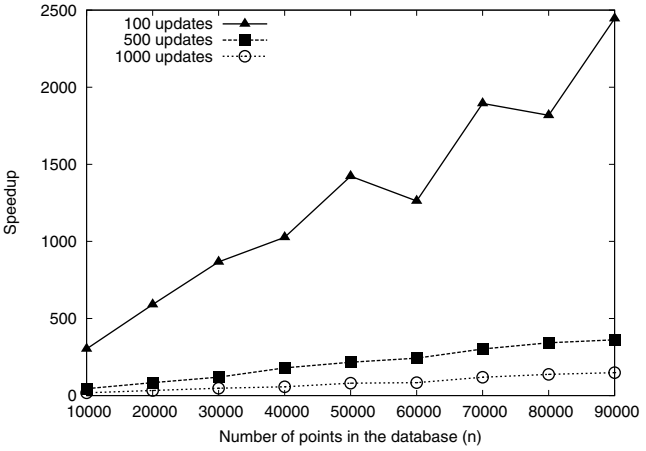

Fig. 15. Speedup achieved by the dynamic algorithm for deletions operations for $Dset_{online}$.

monotonously with the difference between the size of the database and the number of updates because the time complexity of the algorithm also depends on the impact deletions impose on the clustering result.

The final benchmark problem considered, $Dset_{eq}$, is a two dimensional dataset of the longitudes and latitudes of the earthquakes with a magnitude greater than 4, in the Richter earthquake scale, that occurred in the period 1983 to 2003 in Greece. The dataset was obtained from the Institute of Geodynamics of the National observatory of Athens [12]. This dataset is employed

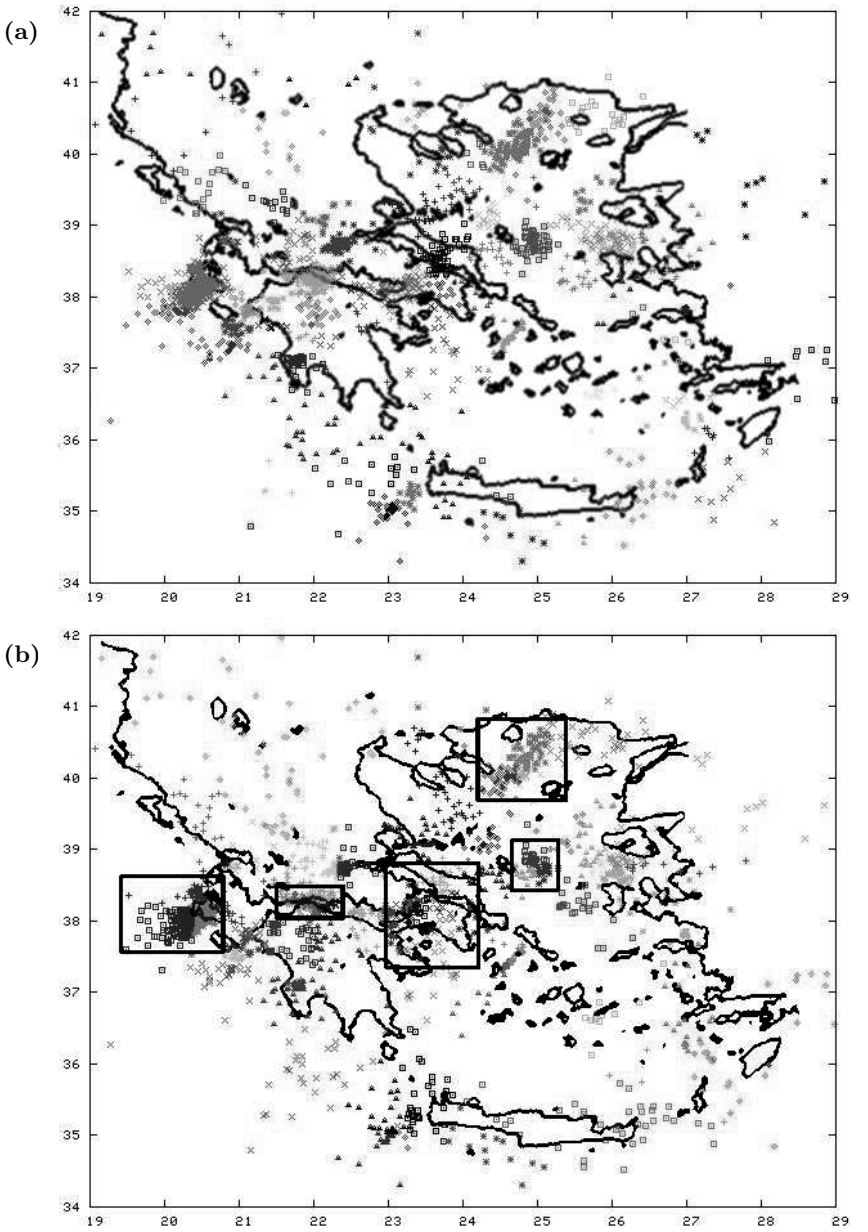

Fig. 16. (a) Results of the k-windows algorithm for $Dset_{eq}$. (b) Results of the k-windows algorithm using fractal dimension for $Dset_{eq}$.

not to obtain a further insight with respect to the earthquake phenomenon,

but rather to study the applicability of the proposed algorithm to a real world dataset.

Fig. 16(a) illustrates the results of the unsupervised k–windows algorithm for $Dset_{eq}$. In Fig. 16(b) illustrates the results of the same dataset of the k-windows algorithm that uses fractal dimension. The modified algorithm separates regions characterized by different fractal dimension, that were assigned to a single cluster by the original algorithm. In Fig. 16(b) characteristic examples of clusters that were separated by the modified algorithm are enclosed in black squares.

This is a preliminary investigation of the application of clustering algorithms to earthquake data. An exhaustive investigation requires the inclusion of additional parameters like magnitude, depth, and time.

7 Concluding remarks

The unsupervised k-windows algorithm is an iterative clustering technique, that attempts to address efficiently the problem of determining the clusters present in a given dataset, as well as, their number. Our experience indicates that the algorithm's performance appears to be robust. With the incorporation of computational geometry techniques the algorithm achieves a comparatively low time complexity. The algorithm has been successfully applied in numerous applications including bioinformatics [47, 48], medical diagnosis [31, 49], time series prediction [35] and web personalization [41].

Given that the development of efficient distributed clustering algorithms has attracted considerable attention in the past few years, the k-windows algorithm has been designed to be easily extended in distributed computing environments, taking under consideration privacy issues and very slow network connections [50]. For the same kind of distributed computing environments, but under the assumption of a high speed underlying network, a parallel version of the algorithm was investigated [4]. This version is able to achieve considerable speedup in execution time and at the same time attain a linear decrease on the storage space requirements with respect to the number of computer nodes used.

For databases that undergo update operations, a technique was presented that is capable of tracking changes in the cluster model [51]. This technique incorporates a dynamic tree data structure (Bkd-tree) that maintains high space utilization, and excellent query and update performance regardless of the number of updates performed. The experimental results suggest that the algorithm is able to identify the changes in the datasets considered, by only updating its cluster model.

Finally, a modification of the k-windows algorithm was presented that uses the fractal dimension of the underlying clusters in order to partition the dataset [52]. This approach enables the identification of regions with different fractal dimension even within a single cluster. The design and development of

algorithms that can detect clusters within clusters is particularly attractive in numerous applications where further qualitative information is valuable. Examples include time–series analysis, image analysis, medical applications, and signal processing.

References

1. P.K. Agarwal and C.M. Procopiuc. Exact and approximation algorithms for clustering (extended abstract). In *Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 658–667, San Francisco, California, U.S.A., 1998.
2. M.S. Aldenderfer and R.K. Blashfield. *Cluster Analysis*, volume 44 of *Quantitative Applications in the Social Sciences*. SAGE Publications, London, 1984.
3. P. Alevizos. An algorithm for orthogonal range search in $d \geqslant 3$ dimensions. In *Proceedings of the 14th European Workshop on Computational Geometry*. Barcelona, 1998.
4. P. Alevizos, D.K. Tasoulis, and M.N. Vrahatis. Parallelizing the unsupervised k-windows clustering algorithm. In R. Wyrzykowski, editor, *Lecture Notes in Computer Science*, volume 3019, pages 225–232. Springer-Verlag, 2004.
5. M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *ACM SIGMOD Int. Conf. on Management of Data*, pages 49–60, 1999.
6. J. Aslam, K. Pelekhov, and D. Rus. A practical clustering algorithm for static and dynamic information organization. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 51–60, 1999.
7. D. Barbarä and P. Chen. Using the fractal dimension to cluster datasets. In *KDD*, pages 260–264. ACM Press, 2000.
8. R.W. Becker and G.V. Lago. A global optimization algorithm. In *Proceedings of the 8th Allerton Conference on Circuits and Systems Theory*, pages 3–12, 1970.
9. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R^*-tree: An efficient and robust access method for points and rectangles. In *ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, 1990.
10. J.L. Bentley and H.A. Maurer. Efficient worst-case data structures for range searching. *Acta Informatica*, 13:155–168, 1980.
11. F. Can. Incremental clustering for dynamic information processing. *ACM Trans. Inf. Syst.*, 11(2):143–164, 1993.
12. Earthquake Catalogue. http://www.gein.noa.gr/services/cat.html, Institute of Geodynamics, National Observatory of Athens.
13. P.K. Chan and S.J. Stolfo. Sharing learned models among remote database partitions by local meta-learning. In *Knowledge Discovery and Data Mining*, pages 2–7, 1996.
14. M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.
15. B. Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.
16. B. Chazelle and L.J. Guibas. Fractional cascading: II applications. *Algorithmica*, 1:163–191, 1986.

17. D. W.L. Cheung, S.D. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In *Database Systems for Advanced Applications*, pages 185–194, 1997.
18. I.S. Dhillon and D.S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence*, pages 245–260, 2000.
19. I.S. Dhillon and D.S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1/2):143–175, 2001.
20. M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *24rd Int. Conf. erence on Very Large Data Bases*, pages 323–333. Morgan Kaufmann Publishers Inc., 1998.
21. M. Ester and R. Wittmann. Incremental generalization for mining in a data warehousing environment. In *Proceedings of the 6th Int. Conf. Extending Database Technology*, pages 135–149. Springer-Verlag, 1998.
22. U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. *Advances in Knowledge Discovery and Data Mining*. MIT Press, 1996.
23. T. Feder D.H. Greene. Optimal algorithm for approximate clustering. In *20th Annual ACM Sympos. Theory Comput.*, pages 434–444, 1988.
24. J.A. Hartigan and M.A. Wong. A k-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.
25. H. Kargupta, W. Huang, K. Sivakumar, and E.L. Johnson. Distributed clustering using collective principal component analysis. *Knowledge and Information Systems*, 3(4):422–448, 2001.
26. KDD. Cup data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999.
27. H-P. Kriegel, P. Kroger, and I. Gotlibovich. Incremental optics: Efficient computation of updates in a hierarchical cluster ordering. In *5th Int. Conf. on Data Warehousing and Knowledge Discovery*, 2003.
28. W. Lam and A.M. Segre. Distributed data mining of probabilistic knowledge. In *Proceedings of the 17th Int. Conf. on Distributed Computing Systems, Washington*, pages 178–185. IEEE Computer Society Press, 1997.
29. L.S. Liebovitch and T. Toth. A fast algorithm to determine fractal dimensions by box counting. *Physics Letters*, 141A(8), 1989.
30. C. Linnaeus. *Clavis Classium in Systemate Phytologorum in Bibliotheca Botanica*. Amsterdam, The Netherlands: Biblioteca Botanica, 1736.
31. G.D. Magoulas, V.P. Plagianakos, D.K. Tasoulis, and M.N. Vrahatis. Tumor detection in colonoscopy using the unsupervised k-windows clustering algorithm and neural networks. In *Fourth European Symposium on "Biomedical Engineering"*, 2004.
32. B. B. Mandelbrot. *The Fractal Geometry of Nature*. Freeman, New York, 1983.
33. N. Megiddo and K.J. Supowit. On the complexity of some common geometric problems. *SIAM Journal on Computing*, 13:182–196, 1984.
34. O. Nasraoui and C. Rojas. From static to dynamic web usage mining: Towards scalable profiling and personalization with evolutionary computation. In *Workshop on Information Technology Rabat, Morocco*, 2003.
35. N.G. Pavlidis, D.K. Tasoulis, and M.N. Vrahatis. Financial forecasting through unsupervised clustering and evolutionary trained neural networks. In *Congress on Evolutionary Computation*, pages 2314–2321, Canberra Australia, 2003.

36. A.P. Pentland. Fractal-based description of natural scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):661–674, 1984.
37. M.G.P. Prasad, S. Dube, and K. Sridharan. An efficient fractals-based algorithm for clustering. In *IEEE Region 10 Conference on Convergent Technologies For The Asia-Pacific*, 2003.
38. F. Preparata and M. Shamos. *Computational Geometry*. Springer Verlag, New York, Berlin, 1985.
39. O. Procopiuc, P.K. Agarwal, L. Arge, and J.S. Vitter. Bkd-tree: A dynamic scalable kd-tree. In T. Hadzilacos, Y. Manolopoulos, and J.F. Roddick, editors, *Advances in Spatial and Temporal Databases, SSTD*, volume 2750 of *Lecture Notes in Computer Science*, pages 46–65. Springer, 2003.
40. V. Ramasubramanian and K. Paliwal. Fast k-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding. *IEEE Transactions on Signal Processing*, 40(3):518–531, 1992.
41. M. Rigou, S. Sirmakessis, and A. Tsakalidis. A computational geometry approach to web personalization. In *IEEE Int. Conf. on E-Commerce Technology (CEC'04)*, pages 377–380, San Diego, California, 2004.
42. J.T. Robinson. The K-D-B-tree: A search structure for large multidimensional dynamic indexes. In *ACM SIGMOD Int. Conf. on Management of Data*, pages 10–18, 1981.
43. J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.
44. N. Sarkar and B.B. Chaudhuri. An efficient approach to estimate fractal dimension of textural images. *Pattern Recognition*, 25(9):1035–1041, 1992.
45. S. Sirmakessis, editor. *Text Mining and its Applications*, volume 138 of *Studies in Fuzziness and Soft Computing*. Springer, 2004.
46. D.K. Tasoulis, P. Alevizos, B. Boutsinas, and M.N. Vrahatis. Parallel unsupervised k-windows: an efficient parallel clustering algorithm. In V. Malyshkin, editor, *Lecture Notes in Computer Science*, volume 2763, pages 336–344. Springer-Verlag, 2003.
47. D.K. Tasoulis, V.P. Plagianakos, and M.N. Vrahatis. Unsupervised cluster analysis in bioinformatics. In *Fourth European Symposium on "Biomedical Engineering"*, 2004.
48. D.K. Tasoulis, V.P. Plagianakos, and M.N. Vrahatis. Unsupervised clustering of bioinformatics data. In *European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems, Eunite*, pages 47–53, 2004.
49. D.K. Tasoulis, L. Vladutu, V.P. Plagianakos, A. Bezerianos, and M.N. Vrahatis. On-line neural network training for automatic ischemia episode detection. In Leszek Rutkowski, Jörg H. Siekmann, Ryszard Tadeusiewicz, and Lotfi A. Zadeh, editors, *Lecture Notes in Computer Science*, volume 2070, pages 1062–1068. Springer-Verlag, 2003.
50. D.K. Tasoulis and M.N. Vrahatis. Unsupervised distributed clustering. In *IASTED Int. Conf. on Parallel and Distributed Computing and Networks*, pages 347–351. Innsbruck, Austria, 2004.
51. D.K. Tasoulis and M.N. Vrahatis. Unsupervised clustering on dynaic databases. *Pattern Recognition Letters*, 2005. in press.
52. D.K. Tasoulis and M.N. Vrahatis. Unsupervised clustering using fractal dimension. *International Journal of Biffurcation and Chaos*, 2005. in press.

53. D.K. Tasoulis and M.N. Vrahatis. Generalizing the k-windows clustering algorithm for metric spaces. *Mathematical and Computer Modelling*, 2005. in press.
54. A. Törn and A. Žilinskas. *Global Optimization*. Springer-Verlag, Berlin, 1989.
55. C. Tryon. *Cluster Analysis*. Ann Arbor, MI: Edward Brothers, 1939.
56. M.N. Vrahatis, B. Boutsinas, P. Alevizos, and G. Pavlides. The new k-windows algorithm for improving the k-means clustering algorithm. *Journal of Complexity*, 18:375–391, 2002.
57. P. Willett. Recent trends in hierarchic document clustering: a critical review. *Inf. Process. Manage.*, 24(5):577–597, 1988.
58. X. Xu, J. Jgerand, and H.P. Kriegel. A fast parallel clustering algorithm for large spatial databases. *Data Mining and Knowledge Discovery*, 3:263–290, 1999.
59. C. Zou, B. Salzberg, and R. Ladin. Back to the future: Dynamic hierarchical clustering. In *Int. Conf. on Data Engineering*, pages 578–587. IEEE Computer Society, 1998.