Available online at www.sciencedirect.com

SCIENCE DIRECT®

Pattern Recognition Letters 26 (2005) 2116–2127

Pattern Recognition Letters

www.elsevier.com/locate/patrec

Unsupervised clustering on dynamic databases

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

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

Received 28 April 2004; received in revised form 19 February 2005
Available online 3 May 2005

Communicated by A. Fred

Abstract

Clustering algorithms typically assume that the available data constitute a random sample from a stationary distribution. As data accumulate over time the underlying process that generates them can change. Thus, the development of algorithms that can extract clustering rules in non-stationary environments is necessary. In this paper, we present an extension of the *k*-windows algorithm that can track the evolution of cluster models in dynamically changing databases, without a significant computational overhead. Experiments show that the *k*-windows algorithm can effectively and efficiently identify the changes on the pattern structure.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Clustering; Pattern recognition; Bkd-tree structure; Dynamic databases

1. Introduction

Clustering can be defined as the process of partitioning a set of patterns into disjoint and homogeneous meaningful groups, called clusters. Clustering is fundamental in knowledge acquisition, and has been applied in numerous fields including, statistical data analysis (Aldenderfer and Blashfield, 1984), compression and vector quantization (Ramasubramanian and Paliwal, 1992), global optimization (Becker and Lago, 1970; Törn and Žilinskas, 1989), and image analysis. It has also been extensively used in social sciences (Aldenderfer and Blashfield, 1984).

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, sometimes radically. A non-stationary environment is characterized by

* Corresponding author. Address: Computational Intelligence Laboratory, Department of Mathematics, University of Patras, GR-26110 Patras, Greece. Tel./fax: +30 2610997348.
E-mail addresses: dtas@math.upatras.gr (D.K. Tasoulis), vrahatis@math.upatras.gr (M.N. Vrahatis).

0167-8655/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2005.03.023

the insertion of new data and the deletion of existing data. In this setting the issue at hand is to update the clustering result at appropriate time intervals with low computational cost. Cluster maintenance deals with this problem. A close examination of clustering algorithms reveals that most of them are unsuitable for this task (Can, 1993). 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 (Zou et al., 1998), web usage user profiling (Nasraoui and Rojas, 2003) as well as, document clustering (Willett, 1988).

From the broader field of data mining, a technique for maintaining association rules in databases that undergo insertions and deletions has been developed in (Cheung et al., 1997). A generalization algorithm for incremental summarization has been proposed in (Ester and Wittmann, 1998). 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 (Charikar et al., 2004). Another on-line star-algorithm for document clustering has been proposed in (Aslam et al., 1999). A desirable feature of this algorithm is that it does not impose any constraint on the number of clusters. Finally, an incremental extension to the GDBSCAN algorithm (Sander et al., 1998) has been proposed in (Ester et al., 1998), along with experimental results for the speedup achieved under a large number of updates. This extension assumes that the parameters of the algorithm remain fixed over time. Using a similar technique an incremental version of the OPTICS algorithm (Ankerst et al., 1999) has been proposed in (Kriegel et al., 2003). The speedup achieved by this incremental algorithm (Kriegel et al., 2003) is significantly lower than that of (Ester et al., 1998). This is attributed to the higher complexity of OPTICS, but Kriegel et al. (2003) claim that the incremental version of OPTICS is suitable for a broader range of applications.

In this paper we propose an extension of the unsupervised k-windows clustering algorithm (Vrahatis et al., 2002) that can efficiently mine clustering rules from databases that undergo insertion and deletion operations over time. The proposed extension incorporates the Bkd-tree structure (Procopiuc et al., 2003). The Bkd-tree is able to efficiently index objects under a significant load of updates, and also provides a mechanism that determines the timing of the updates.

The rest of the paper is organized as follows; in Section 2 we briefly present the unsupervised k-windows clustering algorithm and its computational complexity. Section 3 describes the Bkd-tree structure along with its computational complexity bounds. The proposed modification of the k-windows algorithm is discussed in Section 4 along with a theoretical investigation for the complexity of the proposed approach. Computational experiments are presented in Section 5. The paper ends with concluding remarks.

2. Unsupervised k-windows clustering algorithm

For completeness purposes we briefly discuss the workings of the original k-windows algorithm and its unsupervised extension that automatically approximates the number of clusters. Intuitively, the k-windows algorithm tries to capture all the patterns that belong to one cluster within a d-dimensional window (Vrahatis et al., 2002). To meet this goal it employs two procedures: *movement* and *enlargement*. The movement procedure centers each window at the mean of the patterns it encloses. The movement procedure is iteratively executed, as long as, the distance between the new and the previous centers exceeds the user-defined variability threshold, θ_v. The enlargement process on the other hand, tries to augment the window to include as many patterns from the cluster as possible. To this end, enlargement takes place at each coordinate separately. Each range of a window is enlarged by a proportion θ_e/l, where θ_e is user-defined and l stands for the number of previous successful enlargements. To consider an enlargement successful firstly the movement procedure is invoked. After movement terminates the proportional increase in the number of patterns included in the window is calculated. If this proportional increase exceeds the user-defined coverage threshold, θ_c, then the enlargement is considered successful. In the case that, enlargement was successful for coordinate $c' \geqslant 2$, then all coordinates c'', with $c'' < c'$, 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. The movement and enlargement processes are illustrated in Fig. 1.

The determination of the number of clusters present in a dataset is a fundamental and unresolved issue in cluster analysis. Well-known and widely used iterative techniques, such as the k-means algorithm (Hartigan and Wong, 1979), require from the user to specify the number of clusters present in the data prior to the execution of the algorithm. The unsupervised k-windows algorithm generalizes the original algorithm by approximating the number of clusters at a final stage of the algorithm. The key idea to attain this goal is to apply the k-windows algorithm using a "sufficiently" large number of initial windows and then identify windows that capture patterns belonging to a single cluster. 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, overlapping windows are considered for *merging*. The merge operation is guided by two thresholds the *merge threshold*, θ_m, and the *similarity threshold*, θ_s. Having identified two 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 the mean of these two proportions exceeds θ_s, then the two windows are considered to be identical and the one containing the smaller number of points is ignored. Otherwise, if the mean exceeds θ_m, the windows are considered to belong to the same cluster and are merged. This operation is illustrated in Fig. 2. Specifically, in Fig. 2(a), the extent of overlapping between windows W_1 and W_2 exceeds the θ_s threshold, and thus W_1 is ignored. On the other hand, in Fig. 2(b), the extent of overlapping between windows W_3 and W_4 exceeds only the merge threshold so the algorithm considers both to belong to a single cluster, unlike in Fig. 2(c), where windows W_5 and W_6, capture two different clusters. For a comprehensive description of the algorithm and an investigation of its capability to automatically identify the number of clusters present in a dataset, refer to (Alevizos et al., 2002, 2004; Tasoulis and Vrahatis, 2004; Tasoulis et al., 2003).

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 movement, enlargement and merging procedures the algorithm has correctly identified the three clusters.

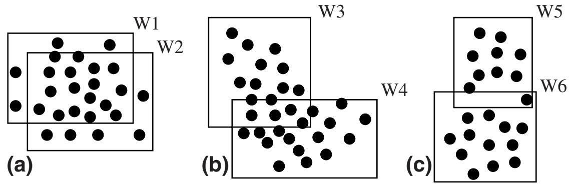

Fig. 2. (a) W_1 and W_2 satisfy the similarity condition and W_1 is ignored. (b) W_3 and W_4 satisfy the merge operation and are considered to belong to the same cluster. (c) W_5 and W_6 have a small overlapment and capture two different clusters.

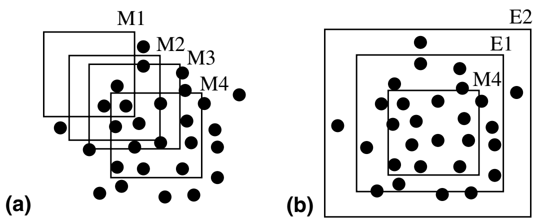

Fig. 1. (a) Sequential movements of the initial window $M1$ that result to the final window $M4$. (b) Sequential enlargements of the initial window $M4$ that result to the final window $E2$.

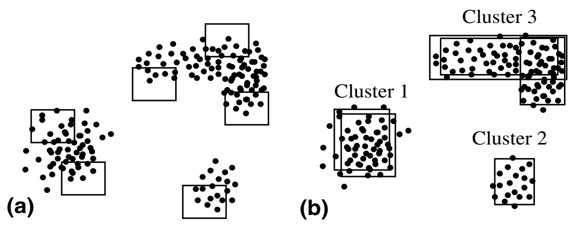

Fig. 3. An example of the application of the k-windows algorithm.

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)

Method	Preprocessing		Query time
	Time	Space	
Preparata and Shamos (1985)	$\theta(dn\log n)$	$\theta(dn)$	$O(s + dn^{1-(1/d)})$
Preparata and Shamos (1985)	$O(n\log^{d-1}n)$	$O(n\log^{d-1}n)$	$O(s + \log^{d}n)$
Preparata and Shamos (1985)	$O(n\log^{d-1}n)$	$O(n\log^{d-1}n)$	$O(s + \log^{d-1}n)$
Chazelle (1986)	$O(n\log^{d-1}n)$	$O\left(n\frac{\log^{d-1}n}{\log\log n}\right)$	$O(s + \log^{d-1}n)$
Chazelle and Guibas (1986)	$O(n\log^{d+1}n)$	$O(n\log^{d}n)$	$O(s + \log^{d-2}n)$
Alevizos (1998)	$O(n\log^{d-1}n)$	$O(n\log^{d-1}n)$	$O(s + \log^{d-2}n)$
Bentley and Maurer (1980)	$O(n^{2d-1})$	$O(n^{2d-1})$	$O(s + d\log n)$
Bentley and Maurer (1980)	$O(n^{1+\varepsilon})$	$O(n^{1+\varepsilon})$	$O(s + \log n)$
Bentley and Maurer (1980)	$O(n\log n)$	$O(n)$	$O(n^{\varepsilon})$

The computationally demanding step of the algorithm is the determination of the points that lie in each d-range. This problem is also referred to as the *range search* problem, and numerous Computational Geometry techniques have been proposed to address it. All these techniques have in common the existence of a preprocessing stage at which they construct a data structure for the patterns. This data structure allows them to answer range queries fast. In Table 1 the computational complexity of various such approaches is summarized.

3. The Bkd-tree

In databases that undergo a significant load of updates, the problem of indexing the data arises. An efficient index should be characterized by high space utilization and small processing time of queries under a continuous updating process. Furthermore, the processing of the updates must be fast. To this end, we employ the Bkd-tree structure proposed in (Procopiuc et al., 2003), that maintains high space utilization and excellent query and update performance, regardless of the number of updates performed.

For completeness purposes we briefly present the Bkd-tree structure. The Bkd-tree is based on a well-known extension of the kd-tree, called the K-D-B-tree (Robinson, 1981), as well as, the so-called logarithmic method for making a static structure dynamic. Extensive experimental studies have shown that the Bkd-tree is able to achieve almost 100% space utilization and also fast query processing of a static K-D-B-tree (Procopiuc et al., 2003). Unlike the K-D-B-tree, however, these properties are maintained under a massive load of updates.

The Bkd-tree structure, instead of maintaining one tree and dynamically re-balancing it after each insertion, it 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. Note that in the expression $\log_2(n/M)$, M stands for the capacity of the memory buffer, in terms of number of points, while n represents the total number of d-dimensional points in the database. To answer a range query using the Bkd-tree, all the $\log_2(n/M)$ trees are queried. Despite this fact, the worst-case behavior of the query time is

$$C = O(dn^{1-(1/d)} + s), \quad (1)$$

where 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 that stores the point and delete it from this tree. Since there are at most $\log_2(n/M)$ trees, the number of operations performed by a deletion is at most $\log(n)\log_2(n/M)$ (Procopiuc et al., 2003).

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, such that T_j is an empty kd-tree, is found. Subsequently, 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 re-organized 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 re-organized.

Extensive experimentation (Procopiuc et al., 2003) 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. Unsupervised *k*-windows on dynamic databases

The proposed dynamic version of the unsupervised *k*-windows algorithm utilizes the Bkd-tree structure. The Bkd-tree is selected because it enables the fast processing of range queries and also provides a criterion for the timing of the update operations on the clustering result. The workings of the dynamic algorithm are outlined in the following framework:

(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.

Through this framework the dynamic algorithm yields a clustering model that adapts to the changes in the database. In the following subsections the treatment of the insertion and deletion operations is thoroughly described along with theoretical results concerning the worst-case time complexity of these operations. The section ends, with a high level description of the overall procedure.

4.1. Treatment of insertions

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 the number of insertions through time causes the T_0 structure to reach the maximum number of points it can store (M) a number of windows is initialized over these points. Subsequently, the movement and enlargement procedures of the unsupervised *k*-windows algorithm are applied on these windows as in the static case. Note that the range queries operate over the entire Bkd-tree, and not only over T_0. After movement and enlargement of the new windows have terminated they are considered for similarity and merging with all the existing windows. Thus, as new windows are processed by the algorithm only the most representative ones are retained, thereby restraining the clustering result to a relatively small size. This procedure is demonstrated in Fig. 4. Specifically, in this figure, the filled circles represent the initial points while the empty circles represent the inserted points. Windows W_1 and W_2 have been finalized from the initial run of the algorithm. Windows W_3 and W_4 are initialized over the inserted points (empty circles). After movement and enlargements operations for W_3 and W_4 terminate, they are considered for similarity and merging. Windows W_1 and W_3 satisfy the merge operation and are considered to belong to the

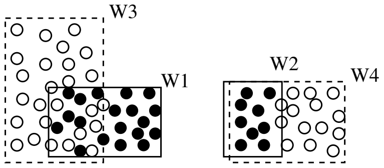

Fig. 4. The application of the *k*-windows algorithm over the inserted points.

same cluster. On the other hand, windows W_2 and W_4 satisfy the similarity condition and W_2 is ignored.

Assuming an upper bound L for the number of range searches that the k-windows algorithm is allowed to perform for the processing of each window, the following lemma can be stated.

Lemma 1. *Let D be the set of the n initial points in the database. Suppose that m insertions of points occur, forming the set D'. Assume further that the unsupervised k-windows algorithm is applied on D using $k = c|D|$, $0 < c \leqslant 1$, initial windows resulting in λ final windows. Then, the dynamic algorithm, applied on $D \cup D'$, using $k' = c|D'|$ initial windows obtained from the set D' only, and considering the λ windows for the merging and similarity operations, achieves a better worst-case time over the static algorithm applied on $D \cup D'$ using $k'' = c|D \cup D'|$ initial windows.*

Proof. Following the above assumptions, and by Eq. (1), the worst-case time performance of the static unsupervised k-windows algorithm, applied on the set $D \cup D'$ is:

$$\begin{aligned} C_s &= \mathrm{O}(Lk''(d|D \cup D'|^{1-(1/d)} + s_1)) \\ &= \mathrm{O}(Lc|D \cup D'|(d|D \cup D'|^{1-(1/d)} + s_1)), \end{aligned} \tag{2}$$

where s_1 is the upper bound of the total retrieved points, which is bounded by the size $|D \cup D'|$ of the dataset. On the other hand, the dynamic k-windows algorithm has a worst run time complexity:

$$\begin{aligned} C_d &= \mathrm{O}(Lk'(d|D'|^{1-(1/d)} + s_2)) \\ &= \mathrm{O}(Lc|D'|(d|D'|^{1-(1/d)} + s_2)), \end{aligned} \tag{3}$$

where s_2 represents the upper bound of the total retrieved points, and is bounded by $|D'|$. Now, since $|D \cup D'| < |D'|$ it is evident that $C_d < C_s$. □

4.2. Treatment of deletions

To address the deletions of points, a second Bkd-tree structure is maintained. Each time a point is deleted, it is removed from the main data structure and is inserted in the second Bkd-tree. When this Bkd-tree, reaches its maximum size, a number of windows is initialized over the points it contains. These windows are subjected to the movement and enlargement procedures of the k-windows algorithm, that operates only on the second database. Subsequently, they 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 all the windows that contain deleted patterns are considered for similarity with the processed ones, the Bkd-tree that stores the deleted points is emptied. In Fig. 5, the deletion process is illustrated. The filled circles represent the points that remain in the database, while the empty circles represent the deleted points. Window W_1 has been finalized from the initial run of the algorithm. Windows W_2 and W_3 are initialized over the deleted points (empty circles). After movement and enlargement, they are considered for similarity with the initial window, W_1. Window W_1 satisfies the similarity condition with W_2 and W_3 and thus it is ignored. Since window W_1 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 W_4, W_5, W_6 and W_7. These windows are considered for merging and similarity. Windows W_4 and W_7 satisfy the similarity operation and thus window W_7 is ignored. Windows W_5 and W_6 satisfy the merge operation thus they are considered to enclose points belonging to the same cluster.

By clustering the deleted objects the dynamic algorithm is able to identify the windows that need to be re-organized in the initial results. Thus, the

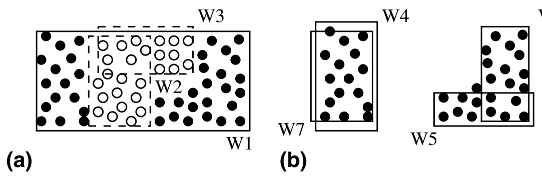

Fig. 5. (a) The application of the k-windows algorithm over the deleted points. (b) The application of the k-windows algorithm over the non-deleted points contained in initial window W_1.

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. Formally, the following lemma can be stated under the assumption of an upper bound, L, on the number of allowable range searches for the processing of each window.

Lemma 2. *Let D be the set of points of the initial n points in the database. Suppose that m deletions of points occur forming the set $D_0 \subset D$, under the restriction that $|D_0| \leqslant 0.5|D|$. Assume further that the k-windows algorithm is applied on the set D using $k = c|D|$, $0 < c \leqslant 1$ initial windows resulting in λ final windows. Suppose further, that the k-windows algorithm applied on the set $D - D_0$ using $k' = c|D - D_0|$ windows, results in λ' final windows, while applying the k-windows algorithm on the set D_0 using $k_0 = c|D_0|$ initial windows results in λ_0 final windows. Then, the dynamic k-windows algorithm achieves a better worst-case time complexity than the static algorithm, as long as, $|D'| \leqslant |D| - 2|D_0|$ where D' is the set that contains the points in D, over which new windows need be initialized.*

Proof. Following the above assumptions, the worst time performance of the static k-windows algorithm, applied on the set $D - D_0$ is:

$$\begin{aligned} C_s &= \mathrm{O}(Lk'(d|D-D_0|^{1-(1/d)} + s_1)) \\ &= \mathrm{O}(Lc|D-D_0|(d|D-D_0|^{1-(1/d)} + s_1)), \end{aligned} \tag{4}$$

where s_1 is the upper bound of the total retrieved points, that is bounded by the size $|D - D_0|$ of the dataset $D - D_0$.

On the other hand, the worst-case time complexity of the dynamic algorithm is:

$$\begin{aligned} C_d &= \mathrm{O}(Lk_0(d|D_0|^{1-(1/d)} + s_2)) \\ &\quad + \mathrm{O}(Lk''(d|D-D_0|^{1-(1/d)} + s_3)) \\ &= \mathrm{O}(Lc|D_0|(d|D_0|^{1-(1/d)} + s_2)) \\ &\quad + \mathrm{O}(Lc|D'|(d|D-D_0|^{1-(1/d)} + s_3)), \end{aligned} \tag{5}$$

where k'' denotes the number of new windows that need to be initialized, over D', and s_2 and s_3, are the upper bounds of the total retrieved points which are bounded by $|D_0|$ and $|D - D_0|$, respectively.

Since $|D_0| \leqslant |D - D_0|$ and $s_2 \leqslant s_3$, we obtain the following bound for C_d:

$$\begin{aligned} C_d &\leqslant \mathrm{O}(Lc|D_0|(d|D-D_0|^{1-(1/d)} + s_2)) \\ &\quad + \mathrm{O}(Lc|D'|(d|D-D_0|^{1-(1/d)} + s_3)) \\ &\leqslant \mathrm{O}(Lc(|D_0| + |D'|)(d|D-D_0|^{1-(1/d)} + s_3)). \end{aligned} \tag{6}$$

Now since it is assumed that, $|D'| \leqslant |D| - 2|D_0|$ we have:

$$|D_0| + |D'| \leqslant |D - D_0|. \tag{7}$$

Using relations (4), (6) and (7), it holds that:

$$\begin{aligned} &\mathrm{O}(Lc(|D_0| + |D'|)(d|D-D_0|^{1-(1/d)} + s_3)) \\ &\quad \leqslant \mathrm{O}(Lc|D-D_0|(d|D-D_0|^{1-(1/d)} + s_1)). \end{aligned} \tag{8}$$

From the above we conclude that $C_d \leqslant C_s$. □

4.3. Proposed algorithm

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

Dynamic unsupervised *k*-windows

```
01. Set {the input parameters of k-windows algorithm}.
02. Initialize an empty set W of d-ranges.
03. Each time the T_0 tree of the Bkd-tree structure is full:
04.     Initialize a set I of k d-ranges, over the T_0 tree.
05.     Perform movements and enlargements of the d-ranges in I.
06.     Update W to contain the resulting d-ranges.
07.     Perform merging and similarity operations of the d-ranges in W.
08. If a large enough number of deletions has been performed:
09.     Initialize a set D of k d-ranges over the deleted points.
10.     Apply k-windows on the d-ranges in D.
11.     If any windows in D satisfy the similarity condition with windows in W:
```

12. **Then** delete those windows from W and
13. **If** the deleted windows from W contained any not deleted points, apply k-windows over them.
14. **Report** the groups of d-ranges that comprise the final clusters.

Using the previously stated lemmata, we formulate the following theorem for the worst-case time complexity of the aforementioned algorithm:

Theorem 3. *Suppose that the assumptions of Lemmata* 1 *and* 2 *are satisfied. Then the dynamic algorithm achieves a better worst-case time complexity for the insertion and deletion stages over the static algorithm.*

Proof. The proof is obvious by Lemmata 1 and 2. □

An important parameter for the execution of the above scheme is the selection of the size, M, of the T_0 component of Bkd-tree structure. This parameter indirectly determines the timing of the update operation of the database (Procopiuc et al., 2003), which in turn triggers the update of the clustering result. Therefore, when applying the proposed algorithm the value of M 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. Presentation of experimental results

To evaluate the performance of the proposed dynamic unsupervised k-windows algorithm two artificial datasets (Dset1 and Dset2), and one real life dataset (Dset3), were used. Both artificial datasets are two-dimensional to allow the visual inspection of the algorithm's performance. Datasets of this kind have been used for the evaluation of various clustering algorithms (Guha et al., 1998; Karyapis et al., 1999; Sander et al., 1998).

Dset1 contains 1600 points organized in four different clusters of different sizes. The dataset was segmented into four parts, each 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. The clustering results for each step are presented in Fig. 6. In detail, in parts (a) and (b) of the figure, seven clusters are identified, an outcome that appears reasonable by means of

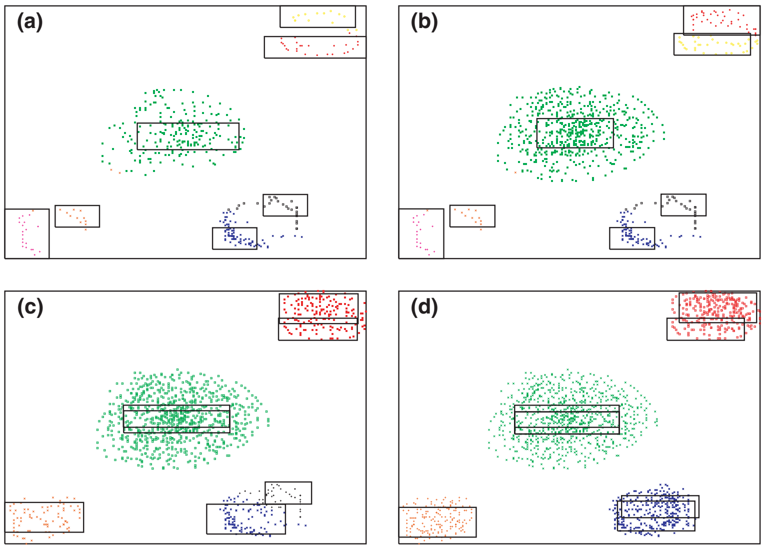

Fig. 6. Applying k-windows into four different instances of Dset1.

visual inspection. In part (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, at step (d) all the clusters are correctly identified, using seven windows.

The second artificial dataset, Dset2, contains 2760 points, organized in three convex and one non-convex cluster. Dset2 was split into five parts. The first part contains 920 points, while the remaining four parts contain 460 points each. When a new part is presented to the algorithm 64 windows are initialized over the new points. The results of the algorithm are depicted in Fig. 7. As it is exhibited in parts (a) and (b), the algorithm correctly identifies two of the clusters present in the dataset. In parts (c)–(e) the algorithm identifies the three convex clusters, as well as, the non-convex one. At the end of the clustering process all four clusters are identified correctly using in total 151 windows. At a next step, 460 points are removed from the database. At this point the algorithm triggers the deletion phase, where the deleted points are removed from the database and are inserted in the second database created for this purpose. Subsequently, 64 windows are initialized over the deleted points and the algorithmic procedure of *k*-windows is applied. Next, the previously discovered windows are considered for similarity with these windows. After this procedure concludes the algorithm correctly identifies five final clusters which are exhibited in part (f) of Fig. 7.

Dataset Dset3 is a part of the KDD 1999 Cup dataset (KDD, 1999). This dataset was generated by the 1998 DARPA Intrusion Detection Evaluation Program that was prepared and managed by the MIT Lincoln Labs. The objective was to

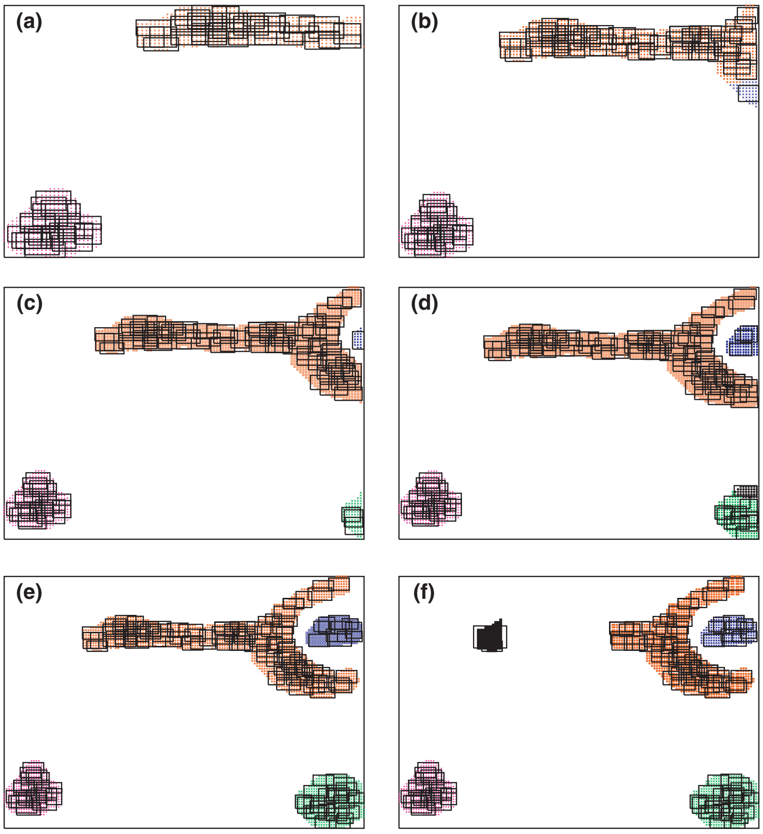

Fig. 7. Applying *k*-windows into six different instances of Dset2.

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 first 100000 records of KDD 1999 Cup train dataset were used. This part of the dataset contains 77888 patterns of normal connections and 22112 patterns of Denial of Service (DoS) attacks. Out of the 42 features the 37 numeric ones were considered. The dataset was randomly permutated, and then split into 10 partitions of 10000 points each. Partitions were presented to the algorithm one at a time. To evaluate the results of the algorithm we will compare them with those obtained by applying k-windows to the entire dataset. 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. On the other hand, applying the dynamic version at each partition of the dataset the algorithm yielded 22 clusters. For each partition 8 windows are initialized. From the 22 clusters, one contained 22087 DoS patterns while the other clusters contained normal patterns, with the exception of 24 DoS patterns that were assigned to four different clusters, with 17, 5, 2 and 1 patterns, respectively. Thus, the dynamic k-windows algorithm produced results of the same quality as the static algorithm.

For comparative purposes with the work of Kriegel et al. (2003) and Sander et al. (1998), we also calculated the speedup achieved by the dynamic version of the algorithm. To this end, we constructed a 10-dimensional dataset, Dset4, 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. 8 and 9. For

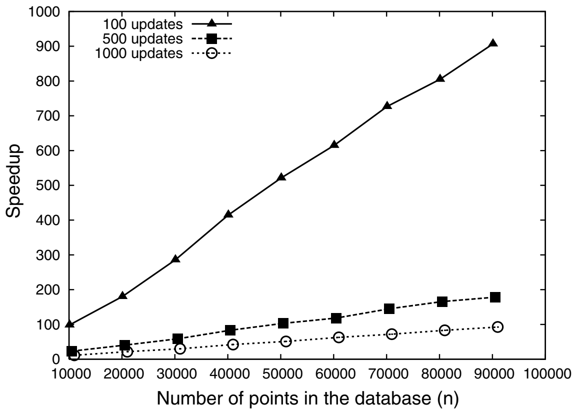

Fig. 8. Speedup achieved by the dynamic algorithm for insertion operations for Dset4.

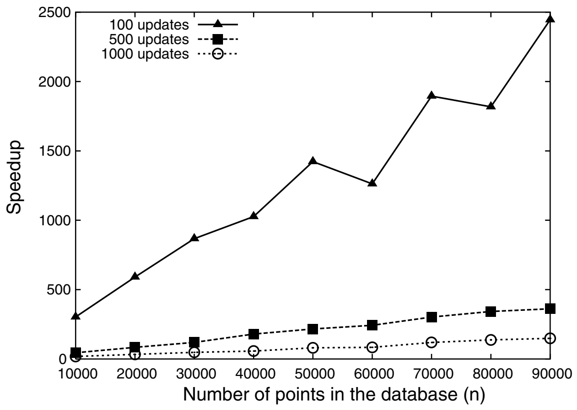

Fig. 9. Speedup achieved by the dynamic algorithm for deletions operations for Dset4.

the insertions case (Fig. 8) 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. 9) 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 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.

6. Concluding remarks

Databases, nowadays, accumulate data at an astonishing rate and over extensive periods of time. These facts generate new challenges for data mining. One such challenge is to develop algorithms that can track changes that occur incrementally. In this paper we present a technique capable of tracking changes in cluster models. Our technique extends the unsupervised *k*-windows clustering algorithm (Vrahatis et al., 2002), and 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 on it. For the proposed approach we provide a proof for the lower worst-case time complexity achieved. Moreover, in the experimental results we firstly illustrate that the algorithm by only updating its cluster model was able to identify the changes in datasets in cases of insertions and deletions. Secondly, measurements of the achieved speedup, indicate the high efficiency of the algorithm, that is on par with other similar approaches (Ester et al., 1998; Kriegel et al., 2003).

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

Aldenderfer, M., Blashfield, R., 1984. Cluster analysis. Quantitative Applications in the Social Sciences, vol. 44. SAGE Publications, London.

Alevizos, P., 1998. An algorithm for orthogonal range search in $d \geqslant 3$ dimensions. In: Proc. 14th European Workshop on Computational Geometry, Barcelona.

Alevizos, P., Boutsinas, B., Tasoulis, D.K., Vrahatis, M.N., 2002. Improving the orthogonal range search *k*-windows clustering algorithm. In: Proc. 14th IEEE Internat. Conf. on Tools with Artificial Intelligence, Washington, DC, pp. 239–245.

Alevizos, P., Tasoulis, D.K., Vrahatis, M.N., 2004. Parallelizing the unsupervised *k*-windows clustering algorithm. In: Wyrzykowski, R. (Ed.), Lecture Notes in Computer Science, vol. 3019. Springer-Verlag, pp. 225–232.

Ankerst, M., Breunig, M., Kriegel, H.-P., Sander, J., 1999. Optics: Ordering points to identify the clustering structure. In: ACM SIGMOD Internat. Conf. on Management of Data (SIGMOD 99), pp. 49–60.

Aslam, J., Pelekhov, K., Rus, D., 1999. A practical clustering algorithm for static and dynamic information organization. In: ACM-SIAM Sympos. on Discrete Algorithms, pp. 51–60.

Becker, R., Lago, G., 1970. A global optimization algorithm. In: Proc. 8th Allerton Conf. on Circuits and Systems Theory, pp. 3–12.

Bentley, J., Maurer, H., 1980. Efficient worst-case data structures for range searching. Acta Inform. 13, 155–168.

Can, F., 1993. Incremental clustering for dynamic information processing. ACM Trans. Inf. Syst. 11 (2), 143–164.

Charikar, M., Chekuri, C., Feder, T., Motwani, R., 2004. Incremental clustering and dynamic information retrieval. SIAM J. Comput. 33 (6), 1417–1440.

Chazelle, B., 1986. Filtering search: A new approach to query-answering. SIAM J. Comput. 15 (3), 703–724.

Chazelle, B., Guibas, L., 1986. Fractional cascading II: Applications. Algorithmica 1, 163–191.

Cheung, D.W., Lee, S., Kao, B., 1997. A general incremental technique for maintaining discovered association rules. In: Database Systems for Advanced Applications, pp. 185–194.

Ester, M., Wittmann, R., 1998. Incremental generalization for mining in a data warehousing environment. In: Proc. 6th Internat. Conf. on Extending Database Technology. Springer-Verlag, pp. 135–149.

Ester, M., Kriegel, H., Sander, J., Wimmer, M., Xu, X., 1998. Incremental clustering for mining in a data warehousing environment. In: 24th Internat. Conf. on Very Large Data Bases. Morgan Kaufmann Publishers, pp. 323–333.

Guha, S., Rastogi, R., Shim, K., 1998. CURE: An efficient algorithm for clustering large databases. In: Proc. ACM-SIGMOD 1998 Internat. Conf. on Management of Data, Seattle, pp. 73–84.

Hartigan, J., Wong, M., 1979. A *k*-means clustering algorithm. Appl. Statist. 28, 100–108.

Karyapis, G., Han, E., Kumar, V., 1999. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. IEEE Comput. 32 (8), 68–75.

KDD, 1999. Cup data. Available from: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>.

Kriegel, H.-P., Kroger, P., Gotlibovich, I., 2003. Incremental optics: Efficient computation of updates in a hierarchical cluster ordering. In: 5th Internat. Conf. on Data Warehousing and Knowledge Discovery.

Nasraoui, O., Rojas, C., 2003. From static to dynamic web usage mining: Towards scalable profiling and personalization with evolutionary computation. In: Workshop on Information Technology Rabat, Morocco.

Preparata, F., Shamos, M., 1985. Computational Geometry. Springer-Verlag, New York.

Procopiuc, O., Agarwal, P., Arge, L., Vitter, J., 2003. Bkd-tree: A dynamic scalable kd-tree. In: Hadzilacos, T., Manolopoulos, Y., Roddick, J. (Eds.), Advances in Spatial and Temporal Databases, SSTD, Lecture Notes in Computer Science, vol. 2750. Springer, pp. 46–65.

Ramasubramanian, V., Paliwal, K., 1992. Fast *k*-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding. IEEE Trans. Signal Process. 40 (3), 518–531.

Robinson, J., 1981. The K-D-B-tree: A search structure for large multidimensional dynamic indexes. In: SIGMOD Internat. Conf. on Management of Data, pp. 10–18.

Sander, J., Ester, M., Kriegel, H.-P., Xu, X., 1998. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. Data Min. Knowl. Disc. 2 (2), 169–194.

Tasoulis, D.K., Vrahatis, M.N., 2004. Unsupervised distributed clustering. In: IASTED Internat. Conf. on Parallel and Distributed Computing and Networks, Innsbruck, Austria, pp. 347–351.

Tasoulis, D.K., Alevizos, P., Boutsinas, B., Vrahatis, M.N., 2003. Parallel unsupervised *k*-windows: An efficient parallel clustering algorithm. In: Malyshkin, V. (Ed.), Lecture Notes in Computer Science, vol. 2763. Springer-Verlag, pp. 336–344.

Törn, A., Žilinskas, A., 1989. Global Optimization. Springer-Verlag, Berlin.

Vrahatis, M.N., Boutsinas, B., Alevizos, P., Pavlides, G., 2002. The new *k*-windows algorithm for improving the *k*-means clustering algorithm. J. Complex. 18, 375–391.

Willett, P., 1988. Recent trends in hierarchic document clustering: A critical review. Inf. Process. Manage. 24 (5), 577–597.

Zou, C., Salzberg, B., Ladin, R., 1998. Back to the future: Dynamic hierarchical clustering. In: International Conference on Data Engineering. IEEE Computer Society, pp. 578–587.