Pattern Recognition 46 (2013) 3040–3055

Contents lists available at SciVerse ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

An algorithm based on density and compactness for dynamic overlapping clustering

Airel Pérez-Suárez [a,b,*], José Fco. Martínez-Trinidad [b], Jesús A. Carrasco-Ochoa [a], José E. Medina-Pagola [a]

[a] National Institute of Astrophysics, Optics and Electronics (INAOE), Puebla, Mexico
[b] Advanced Technologies Application Center (CENATAV), Havana, Cuba

ARTICLE INFO

Article history:
Received 5 June 2012
Received in revised form
25 March 2013
Accepted 27 March 2013
Available online 10 April 2013

Keywords:
Data mining
Clustering
Overlapping clustering algorithms
Dynamic clustering algorithms

ABSTRACT

Most clustering algorithms organize a collection of objects into a set of disjoint clusters. Although this approach has been successfully applied in unsupervised learning, there are several applications where objects could belong to more than one cluster. Overlapping clustering is an alternative in those contexts like social network analysis, information retrieval and bioinformatics, among other problems where non-disjoint clusters appear. In addition, there are environments where the collection changes frequently and the clustering must be updated; however, most of the existing overlapping clustering algorithms are not able to efficiently update the clustering. In this paper, we introduce a new overlapping clustering algorithm, called DClustR, which is based on the graph theory approach and it introduces a new strategy for building more accurate overlapping clusters than those built by state-of-the-art algorithms. Moreover, our algorithm introduces a new strategy for efficiently updating the clustering when the collection changes. The experimentation conducted over several standard collections shows the good performance of the proposed algorithm, wrt. accuracy and efficiency.

1. Introduction

Clustering is a fundamental machine learning and data mining task. It aims at grouping a set of objects into a set of classes called *clusters*, such that objects belonging to the same cluster are similar enough to infer they are of the same type and, objects belonging to different clusters are different enough to infer they are of different types [1]. There are several examples of the application of clustering algorithms in different areas: medicine [2,3], topic detection and tracking [4], image segmentation [5], social network analysis [6], biology [7] and image classification [8], among others.

Most of the clustering algorithms that have been proposed do not allow objects to belong to more than one cluster, i.e., they build disjoint clusters. Although this approach has been successfully applied in unsupervised learning, there are several situations in which a richer model is needed for representing the data. For example, a patient at a health care facility could have more than one disease. Therefore, if we cluster the set of patients according to their symptoms, it could be expected that some patients belong to more than one cluster. There are other applications like text segmentation [9], information retrieval [10] and news stream analysis [11], among others, where objects could belong to more than one cluster. For this kind of applications, overlapping clustering is useful and important.

Several clustering algorithms, addressing the problem of overlapping clustering, have been reported in the literature; nevertheless, there are environments like the World Wide Web, news streams and others, which impose an additional requirement for these algorithms. In these environments the collection changes frequently; thus, the algorithms should be able to efficiently update the clustering after changes. The capability of processing changes over an already clustered collection is commonly used as a criterion for classifying the clustering algorithms. According to this criterion a clustering algorithm can be classified as *static*, *incremental* or *dynamic*.

Static algorithms suppose that the entire collection is available before clustering; therefore, when objects are added to or deleted from the collection, these algorithms rebuild the clusters starting from scratch by reprocessing the whole collection, i.e., they do not take advantage of the previously built clusters for updating the clustering after changes. On the other hand, *incremental algorithms* are able to process new objects added to the collection and, consequently, they can update the clustering using the previous clusters. Finally, *dynamic algorithms* are able to update the clustering when some objects are added, removed or modified. A modification is usually seen as a deletion followed by an addition and it will be considered as such in this work.

From the overlapping clustering algorithms reported in the literature, only a few are able to process changes in the data collection. For this reason, the problem of dynamic overlapping clustering is addressed in this work.

* Corresponding author at: Advanced Technologies Application Center (CENATAV), Havana, Cuba. Tel.: +537 272 1676; fax: +537 273 0045.
E-mail addresses: airel@ccc.inaoep.mx, airel26@gmail.com, asuarez@cenatav.co.cu (A. Pérez-Suárez), fmartine@inaoep.mx (J.Fco. Martínez-Trinidad), ariel@inaoep.mx (J.A. Carrasco-Ochoa), jmedina@cenatav.co.cu (J.E. Medina-Pagola).

0031-3203/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.patcog.2013.03.022

The main contribution of this paper is a new dynamic overlapping clustering algorithm based on graph theory. The proposed algorithm, called DClustR (Dynamic Overlapping Clustering based on Relevance), introduces a new graph-covering strategy and a new filtering strategy, which together allow to obtain a small set of overlapping clusters. Additionally, DClustR introduces a new strategy for efficiently updating the clustering after multiple object additions and/or deletions. The above characteristics make DClustR suitable for handling overlapping clustering in applications where the object collection changes frequently, specially for those applications handling multiple changes at the same time.

Our experimental evaluation, conducted over several standard data collections, shows that the proposed algorithm outperforms in terms of quality, according to the FBcubed evaluation measure [12], the overlapping clustering algorithms of the state-of-the-art that are able to process changes in a collection. Moreover, the experimental results show that DClustR offers a better trade off between quality and efficiency than the previous incremental and dynamic overlapping clustering algorithms.

The rest of this paper is structured as follows: Section 2 describes related work. In Section 3, we introduce the DClustR algorithm. An experimental evaluation, showing the performance of our proposed algorithm on several data collections, is presented in Section 4. Finally, conclusions and future work are discussed in Section 5.

2. Related work

The algorithm proposed in this work is related with two research areas. The first one is incremental or dynamic clustering, and the second one is overlapping clustering. Although algorithms proposed for incremental fuzzy clustering, like those introduced in [13], are related with our work, there is an important aspect we would like to highlight. Unlike the algorithms in [13] where fuzzy memberships to clusters are used to represent the fact that an object can belong to more than one cluster, in the proposed algorithm crisp memberships to clusters are computed; therefore, a direct comparison of the proposed algorithm against this kind of algorithms is not possible. Extending our algorithm to produce fuzzy clusterings and a comparison against incremental fuzzy clustering algorithms are out of the scope of this paper. Therefore, this type of algorithms will not be analyzed in this section.

There have been several works addressing incremental or dynamic clustering [14–17]; however, most of them build disjoint clusters. On the other hand, there are several clustering algorithms proposed for overlapping clustering, however, most of these algorithms are static; therefore, they cannot efficiently process additions and/or deletions of objects.

Although our work seems to be close to algorithms developed for clustering data streams [18–20], there are some differences that must be highlighted. First, since a data stream consists of a set of multi-dimensional records $X_1, X_2, \ldots, X_k, \ldots$ arriving at different time stamps, those algorithms proposed for clustering data streams, for example [18–20], are just able to process additions. On the other hand, dynamic algorithms, like the one we introduce in this work, are able to process additions, deletions and modifications. Second, to the best of our knowledge, none of the algorithms developed for clustering data stream addresses the problem of overlapping clustering, which is the problem we are analyzing in this work. Based on these differences, we decided not to include as related work algorithms addressing the data stream clustering problem.

Algorithms developed for evolutionary clustering [21–23], also seem to be close to our work; however, there are some important differences. First of all, dynamic clustering algorithms focus on how to modify a current clustering to take into account additions, deletions and modifications in the collection, in order to provide an approximation to the clustering that we would get if the whole collection were clustered starting from scratch. On the other hand, evolutionary clustering algorithms focus on how to mine the collection for finding interpretable cluster evolutions. Moreover, to the best of our knowledge, none of the algorithms developed for evolutionary clustering addresses the problem of overlapping clustering, which is the problem we are analyzing in this work. Therefore, algorithms facing the evolutionary clustering problem were not included as related work.

Based on all the above, we briefly review the algorithms which are able to both produce overlapping clusterings and to update the current clustering when the collection changes.

The overlapping clustering algorithms that can deal with changes in the collection are: Star [10], ISC (Incremental Strong Component) [11], STC (Suffix Tree Clustering) [24], SHC (Similarity Histogram Clustering) [25], ICSD (Incremental Clustering based on Strength Decision) [26], DCS (Dynamic Clustering based on Strength) [27], DHS (Dynamic Hierarchical Star) [28] and the algorithm of Duan et al. [29]. The algorithm of Duan et al., DCS, Star and DHS are dynamic algorithms while the others are incremental algorithms, i.e., they can deal only with additions. On the other hand, DHS is a hierarchical algorithm while the others are non-hierarchical. It is important to mention that DCS and ICSD use the same strategy for clustering but ICSD can deal only with additions while DCS can process both additions and deletions.

Star [10], ISC [11], ICSD [26] and DCS [27] are *graph-based* clustering algorithms which represent the collection of objects as a similarity graph. A similarity graph, denoted as G, is an undirected graph in which vertices represent objects of the collection and edges represent similarity relations among objects.

Star, ICSD and DCS represent the collection as a *thresholded similarity graph*, while ISC represents the collection as a *maximal thresholded similarity graph*. A *thresholded similarity graph*, denoted as G_β, is a similarity graph such that there is an edge between two objects iff their similarity is greater than or equal to a predefined threshold β. On the other hand, a *maximal thresholded similarity graph*, denoted as $G_{max-\beta}$, is a directed similarity graph whose vertices represent objects of the collection and there is a directed edge from object v to object u iff the similarity of v wrt. u is the highest among all the objects. These three algorithms build a clustering through a covering of the graph they use for representing the collection, i.e., a vertex covering. For covering G_β, Star, ICSD and DCS use a special kind of sub-graph, called *star-shaped sub-graph* (s-graph); for covering $G_{max-\beta}$, the ISC algorithm uses *strong compact sets* (sc-set).

A *star-shaped subgraph* (s-graph) in G_β is a subgraph of G_β such that there is a vertex which has an edge with any other vertex in the subgraph. On the other hand, a *strong compact sets* (sc-set) is a minimal subset of objects such that any object, belonging to the set, also has in the set all its more similar objects. Once the covering is built, each s-graph or sc-set is interpreted as a cluster. The computational complexity of Star is $O(n^2 \cdot \log^2 n)$. On the other hand, the computational complexity of ISC, ICSD and DCS is $O(n^2)$.

These four algorithms have several limitations. First of all, as we will demonstrate in our experiments, they build clusterings having a large number of clusters. Usually, the number of clusters is unknown a priori in real problems. However, when we use a clustering algorithm in order to discover hidden relations among objects in a collection, the number of clusters should be small wrt. the number of objects in the collection. Note that, if the number of clusters grows, analyzing those clusters could be as difficult as analyzing the entire collection.

Another limitation of Star, ICSD and DCS, as we will show in our experiments, is that they build clusterings with high overlapping.

As it was mentioned in the Introduction, to obtain overlapping clusters is very useful for several applications; however, when the overlapping among the clusters is too high, it could be difficult to obtain useful findings about the structure of the data. Besides, there are applications like document segmentation by topic, where a high overlapping could be a signal of a bad segmentation [9]. A similar example can be also found in the context of social networks analysis [30,31].

Finally, although Star can process both additions and deletions of objects, it cannot process more than one change at the same time, i.e., Star must process those changes one by one in order to update the clustering.

It is important to clarify what we understand as *multiple* additions/deletions and why it is important to deal with them. Let us suppose that a set of objects O will be added to an already clustered collection. For processing these changes, Star adds the objects of O one by one, updating the clustering after each addition. Notice that, if more than one object in O affects the same cluster, then this strategy could be very expensive because the same cluster would be updated several times. A better choice is to allow the algorithm adding all the objects to the collection and, after that, updating the clusters affected by those additions. This situation also happens with deletions or with a combination of additions and deletions.

Another overlapping clustering algorithm able to update the clustering when the collection changes is STC [24]. This algorithm is incremental and it was developed for clustering collections of *snippets*. Snippets are small texts used by systems like Google to describe the results of a web search. For building the clustering, STC builds a *suffix tree* [32] containing all the suffixes of all the snippets in the collection. A *suffix tree* is an acyclic connected graph in which each vertex is labeled with a string that is common to the collection of snippets the vertex contains. Then, STC determines which nodes will be used as seeds for building the clusters. Finally, STC builds the clusters by merging some of the previously detected seeds, following a strategy similar to that one used by single-link algorithm [33]. Even though, the authors of STC claim that the computational complexity of STC is $O(n \cdot \log n)$, as it was mentioned in [25], the complexity of STC could get to be exponential, depending on the number of suffixes contained in all the snippets.

The main limitation of STC is related to the construction of the suffix tree. Although the construction of this tree depends on the snippets to cluster, generally, it could be very expensive when the number of snippets grows. Finally, it is important to mention that, although STC can add multiple suffixes to the suffix tree at the same time, STC would consume a long time for updating a clustering because it must rebuild all the clusters starting from scratch, every time a change is processed.

Another overlapping clustering algorithm, able to update the clustering when the collection changes, is SHC [25]. SHC is an incremental clustering algorithm based on the concept of *Histogram Ratio* of a cluster. The histogram ratio of a cluster is a measure of cluster cohesiveness. For clustering an object collection, SHC processes the objects in an incremental way. For each object o, SHC computes the most suitable clusters in which o should be added; for this purpose, SHC analyzes how much the histogram ratio of each cluster would vary if o was added to it. Finally, after checking all clusters, if o was not added to any cluster then o constitutes a new cluster. The computational complexity of SHC is $O(n^2)$.

SHC has several limitations. First, it needs to tune values for several parameters (β, ϵ and HR_{min}), and these values depend on the collection to process. In general, users do not have any a priory knowledge about the collection they want to cluster; therefore, to tune up several parameters could be a difficult task. Second, as we will show in our experiments, SHC builds clusterings with high overlapping. Additionally, as the Star algorithm, SHC is not able to process multiple additions.

Another algorithm able to build an overlapping clustering and to update it when the collection changes is DHS [28]. This algorithm is hierarchical and it is derived from the dynamic hierarchical agglomerative framework proposed in [28]. In the first level of the hierarchy, DHS considers each object of the collection as a cluster and, starting from this point, each level is built using as objects the clusters of the previous level. For building the clustering on each level, DHS uses a strategy comprised of four steps. In the first step, DHS builds a thresholded similarity graph G_β, for representing the collection of objects of the level. After that, if G_β has no edge, the building process stops and all the levels previously built constitute the resulting hierarchy. Otherwise, if G_β has at least one edge, in the third step DHS builds from G_β a maximal thresholded similarity graph $G_{max-\beta}$. Afterwards, in the fourth step DHS builds the clustering through a covering of $G_{max-\beta}$, using a variation of the Star algorithm [10]. There are two main differences between the variation used by DHS and the original Star. First, unlike Star, DHS allows the centers of the s-graphs to be adjacent to each other center. Second, Star builds a covering of a thresholded similarity graph, while DHS builds a cover of an undirected maximum β-similarity graph. The computational complexity of DHS is $O(n^3)$.

DHS has several limitations. First of all, with the aim of speeding up the calculation of the similarity among clusters, useful to build the graph G_β in each level, DHS imposes the following constraints: (i) the objects of the collection must be represented using the vector space model (VSM) [34], (ii) a cluster must be represented using the *composite vector of the cluster*; i.e., the sum of the vectors of all objects in the cluster, and (iii) the cosine measure [35] must be used as similarity measure. These constraints, mainly (i) and (iii), reduce the application scope of DHS. Besides, since overlapping among clusters is allowed, the composite vector of a cluster cannot be built from the composite vectors of the sub-clusters it contains. Thus, this composite vector must be built from the vectors of all the objects contained in the cluster; which increases the processing time of DHS. Another limitation of DHS is that it could leave uncovered objects in any level of the hierarchy.

Another algorithm able to build overlapping clusters and to process changes in the collection is the algorithm proposed by Duan et al. [29], developed for social network analysis; in the following we will refer to this algorithm as Duan's algorithm. This algorithm represents the collection of objects as a graph $G(V, E)$, where V denotes entities in a social network and the edge set E denotes relationships between entities. For building an overlapping set of clusters, this algorithm employs the same idea proposed by Palla et al. in CPM (Clique Percolation Method) [36]. However, the Duan's algorithm has two main limitations. First of all, its computational complexity could get to be exponential in the worst scenario. Second, as the Star algorithm, this algorithm is not able to process multiple changes.

As it was pointed out above, the overlapping clustering algorithms reported in the literature that can deal with changes in the collection have several limitations. In this work, we propose a new overlapping clustering algorithm, called DClustR, which solves the limitations of previous algorithms. Unlike previous works, DClustR is able to efficiently update the clustering when the collection changes due to multiple additions and/or deletions. The strategy that DClustR follows for building the clustering allows to obtain less clusters than those built by state-of-the-art algorithms. Besides, the clusters built by DClustR have less overlapping than those produced by previous algorithms. Moreover, as we will show in our experiments, the clusterings built by DClustR have better quality than those clusterings built by state-of-the-art algorithms.

3. The DClustR clustering algorithm

In this section, a new dynamic clustering algorithm for building overlapping clusters is introduced. The presentation of our algorithm, called DClustR (Dynamic Overlapping Clustering based on Relevance), is divided in three parts. First, in Section 3.1, we give some basic concepts needed for introducing DClustR. In Section 3.2, we explain the ideas used by DClustR for building overlapping clusterings. After that, in Section 3.3, we present the strategy that DClustR follows for updating an overlapping clustering when multiple additions and/or deletions are done over an already clustered collection.

3.1. Basic concepts

Let $O=\{o_1,o_2,\ldots,o_n\}$ be a collection of objects, $\beta\in[0,1]$ a similarity threshold and $S(o_i,o_j)$ a symmetric similarity function.

A *weighted thresholded similarity graph* is an undirected and weighted graph $\tilde{G}_\beta=\langle V,\tilde{E}_\beta,S\rangle$ such that $V=O$, each edge $(v,u)\in\tilde{E}_\beta, v\neq u$ is labeled with the value of $S(v,u)$ and, there is an edge $(v,u)\in\tilde{E}_\beta$ iff $S(v,u)\geq\beta$.

We will say that two vertices $v,u\in V$ are *adjacent* iff there is an edge $(v,u)\in\tilde{E}_\beta$. The set of adjacent vertices of a vertex v will be denoted by $v.Adj$. Besides, the size of the set of adjacent vertices of any vertex v will be known as the *degree* of v; any vertex v having $|v.Adj|=0$ is called *isolated*.

Let $\tilde{G}_\beta=\langle V,\tilde{E}_\beta,S\rangle$ be a weighted thresholded similarity graph. A *weighted star-shaped sub-graph* (*ws-graph*) in $\tilde{G}_\beta$, denoted by $G^\star=\langle V^\star,E^\star,S\rangle$, is a sub-graph of $\tilde{G}_\beta$ having a vertex $c\in V^\star$ such that there is an edge between c and all other vertices in $V^\star$. The vertex c is called the *center* of the ws-graph and the remaining vertices are called *satellites*. Isolated vertices are considered *degenerated* ws-graphs.

Let $\tilde{G}_\beta=\langle V,\tilde{E}_\beta,S\rangle$ be a weighted thresholded similarity graph and $W=\{G_1^\star,G_2^\star,\ldots,G_k^\star\}$ be a set of graphs, such that each $G_i^\star=\langle V_i^\star,E_i^\star,S\rangle$ is a ws-graph in $\tilde{G}_\beta$. The set W is a *covering* of $\tilde{G}_\beta$ iff it meets that $V=\cup_{i=1}^k V_i^\star$. Additionally, we will say that a ws-graph $G_i^\star\in W$ *covers* a vertex v iff $v\in V_i^\star$.

Let $G_v^\star=\langle V_v^\star,E_v^\star,S\rangle$ be a ws-graph, having v as its *center*. The intra-cluster similarity of $G_v^\star$, denoted as $Intra\_Sim(G_v^\star)$, is the average similarity between all pair of vertices belonging to $V^\star$ [37], that is

$$Intra\_Sim(G_v^\star)=\frac{\sum_{z,u\in V_v^\star,z\neq u}S(z,u)}{\frac{|V_v^\star|\cdot(|V_v^\star|-1)}{2}} \tag{1}$$

Computing $Intra\_Sim(G_v^\star)$ using (1) is $O(n^2)$. Thus, using this equation for computing the intra-cluster similarity of the ws-graph determined by all vertices in $\tilde{G}_\beta$ becomes $O(n^3)$. In order to use (1) in our algorithm, we should find an efficient approximation of it.

From the definition of weighted thresholded similarity graph, we have that all edges (x,y) such that $S(x,y)<\beta$ do not belong to $\tilde{G}_\beta$. Since any ws-graph $G_v^\star$ is a sub-graph of $\tilde{G}_\beta$, these edges do not belong either to $G_v^\star$. Therefore, if these edges are not taken into account in the computation of $Intra\_Sim(G_v^\star)$, we could rewrite (1) as follows:

$$Aprox\_Intra\_Sim(G_v^\star)=\frac{\sum_{(z,u)\in E_v^\star}S(z,u)}{\frac{|V_v^\star|\cdot(|V_v^\star|-1)}{2}} \tag{2}$$

Although when we save some calculations, Eq. (2) is still $O(n^2)$. However, it can be noticed from (2) that, for a given ws-graph $G_v^\star$, the value of the numerator of $Aprox\_Intra\_Sim(G_v^\star)$ only depends on the number of edges of $G_v^\star$, as well as on their weights. Therefore, based on the definition of ws-graph, we know that the lowest value of $Aprox\_Intra\_Sim(G_v^\star)$ is reached when the edges in $E_v^\star$ are only those between the center v and the satellites of $G_v^\star$. Assuming this worst scenario, we could simplify (2) as follows:

$$Aprox\_Intra\_Sim(G_v^\star)=\frac{\sum_{u\in V_v^\star,u\neq v}S(v,u)}{\frac{|V_v^\star|\cdot(|V_v^\star|-1)}{2}} \tag{3}$$

Computing $Aprox\_Intra\_Sim(G_v^\star)$ using (3) is $O(n)$. Thus, we can use (3) for computing $Aprox\_Intra\_Sim(G_v^\star)$ for each ws-graph $G_v^\star$ in $\tilde{G}_\beta$, this way we save computational time. However, the denominator of (3) grows faster than its numerator. Therefore, a minor increase in the cardinality of $V^\star$ could represent a big decrease in the value of $Aprox\_Intra\_Sim(G_v^\star)$. Moreover, the above mentioned characteristic of (3) makes the value of $Aprox\_Intra\_Sim(G_v^\star)$ to be near 0, for larger values of $|V_v^\star|$. This could represent a bias for ws-graphs with many satellites (i.e., big ws-graphs) and also, it makes difficult the comparison between values of the *Aprox_Intra_sim* for two big ws-graphs. Based on this, we substitute the denominator of (3) by $(|V_v^\star|-1)$; that is, we approximate the value of $Aprox\_Intra\_Sim(G_v^\star)$ as the average weight of the edges existing in $G_v^\star$ between v and the satellites. Then, we could simplify (3) as follows:

$$Aprox\_Intra\_Sim(G_v^\star)=\frac{\sum_{u\in V_v^\star,u\neq v}S(v,u)}{|V_v^\star|-1} \tag{4}$$

3.2. Building overlapping clusters based on relevance

In this section, we explain the strategy used by DClustR for building a set of overlapping clusters. The main idea of this strategy is to generate an initial set of clusters by covering $\tilde{G}_\beta$ using ws-graphs and, after that, to improve these initial clusters in order to obtain the final clustering. In this context, "to improve a set of clusters" means to reduce both the number of clusters and the overlapping among them.

As it can be seen from the previous section, a ws-graph is determined by its center; thus, the problem of building a set $W=\{G_{c_1}^\star,G_{c_2}^\star,\ldots,G_{c_k}^\star\}$ of ws-graphs, such that W is a covering of $\tilde{G}_\beta$, can be seen as the problem of building a set $X=\{c_1,c_2,\ldots,c_k\}$ such that $c_i\in X$ is the center of $G_{c_i}^\star\in W$, $\forall i=1..k$.

Since each vertex in $\tilde{G}_\beta$ can form a ws-graph, we should analyze all vertices in $V^\star$ for building X. In order to prune the search space and to establish a criterion for selecting the vertices that should be included in X, DClustR introduces the concept of *relevance* of a vertex. For defining the *relevance* of a vertex v, first we will define the *relative density* and the *relative compactness* of a vertex v.

A simple idea for reducing the number of ws-graphs needed for covering $\tilde{G}_\beta$ would be to iteratively select the vertices with highest degree. In this way, we try to maximize the number of vertices that are added to the covering of $\tilde{G}_\beta$ on each iteration. The Star algorithm [10] for building the clustering follows a similar idea for selecting the s-graphs. However, suppose that the vertices $v_1,v_2,\ldots,v_k$ were selected in the first k iterations. Let u be the vertex, among those non-selected vertices, having the highest degree in the iteration $k+1$. If there are d vertices of the k previously selected vertices that are adjacent to u and u is added to X, then only $|u.Adj|-d$ vertices would be added to the covering of $\tilde{G}_\beta$. But, if in the iteration $k+1$ exists another vertex z such that: (i) $|z.Adj|<|u.Adj|$, (ii) there are d_1 of the previously selected vertices that are adjacent to z, and (iii) $|u.Adj|-d<|z.Adj|-d_1$, then selecting vertex z in the iteration $k+1$ would be a better choice than selecting u.

Based on the aforementioned analysis, we can affirm that for any vertex v, the number of adjacent vertices having a degree non-greater than the degree of v, is a good estimation about how many adjacent vertices (potentially uncovered) v could add to the covering of $\tilde{G}_\beta$.

Now, suppose that in the iteration $k+1$, there are two vertices v and u, such that v can add six vertices to the covering and u can add four vertices to the covering. Also suppose that $|v.Adj| = 10$ and $|u.Adj| = 5$. If we make our decision based on the number of vertices that each vertex can add to the covering, the choice would be the vertex v. Nevertheless, vertex u can cover a greater percent of its adjacent vertices than vertex v (4/5 versus 6/10). Motivated by the aforementioned ideas, we introduce the concept of *relative density* of a vertex.

The *relative density* of a vertex $v \in V$, denoted as $v.densityR$, is computed as follows:

$$v.densityR = \frac{v.density}{|v.Adj|}$$

where $v.density$ denotes the number of adjacent vertices of v having a degree non-greater than the degree of v. The density of v expresses how many adjacent vertices (potentially uncovered) could be included by v in the covering of $\tilde{G}_\beta$. The relative density takes values in [0,1] and it determines how relevant is the density of v wrt. the number of adjacent vertices of v. The higher the value of $v.densityR$, the greater the number of adjacent vertices (potentially uncovered) that v could include in the covering of $\tilde{G}_\beta$ and therefore, the better v is for covering $\tilde{G}_\beta$.

It is important to highlight that the vertices that were not counted in $v.density$ are those selected in previous iterations; therefore, a high value of $v.densityR$ also means that the ws-graph determined by v has low overlapping with the previous selected ws-graphs. In this way, we also reduce the overlapping among the selected ws-graphs.

Relative density can be used for reducing the number of ws-graphs needed for covering $\tilde{G}_\beta$ and for selecting ws-graphs with low overlapping. However, since this property is mainly based on the vertex degree, using only the relative density could lead to select ws-graphs with a high number of satellites but with low average similarity among them. For solving this issue, we define the concept of *relative compactness* of a vertex v. This new concept will help us to identify the best vertices for covering $\tilde{G}_\beta$, taking into account the *Aprox_Intra_Sim* of the ws-graph determined by each vertex, instead of its degree as the relative density does.

The *relative compactness* of a vertex $v \in V$, denoted as $v.compactnessR$, is computed as follows:

$$v.compactnessR = \frac{v.compactness}{|v.Adj|}$$

where $v.compactness$ denotes the number of vertices $u \in v.Adj$ such that $Aprox\_Intra\_Sim(G_v^\star) \geq Aprox\_Intra\_Sim(G_u^\star)$, being $G_v^\star$ and $G_u^\star$ the ws-graphs determined by v and u, respectively. The compactness of v expresses, taking into account the *Aprox_Intra_Sim* of the ws-graph determined by each vertex instead of its degree, how many adjacent vertices (potentially uncovered) could be included by v in the covering of $\tilde{G}_\beta$. The relative compactness takes values in [0,1] and it determines how relevant is the compactness of v wrt. the number of adjacent vertices of v. Similar to $v.densityR$, the higher the value of $v.compactnessR$, the greater the number of adjacent vertices (potentially uncovered) that v could include in the covering of $\tilde{G}_\beta$; therefore, the better v is for covering the graph.

Finally, we define the *relevance* of a vertex v, denoted as $v.relevance$, as

$$v.relevance = \frac{v.densityR + v.compactnessR}{2} \tag{5}$$

where $v.densityR$ and $v.compactnessR$ are the relative density and relative compactness of v, respectively. Since both $v.densityR$ and $v.compactnessR$ take values in [0,1], the relevance of a vertex also takes values in [0,1]. From (5) it can be inferred that a high value of relevance will correspond with vertices having high values of $v.densityR$ and/or $v.compactnessR$; that is, the higher the value of $v.relevance$ the better v is for covering $\tilde{G}_\beta$. Therefore, in order to build the covering of $\tilde{G}_\beta$, we must analyze the vertices in descending order according to their relevance. Finally, since vertices having $v.relevance = 0$ are those vertices having $v.densityR = 0$ and $v.compactnessR = 0$, we do not have to analyze vertices having zero relevance. In Fig. 1, we show an example of a weighted thresholded similarity graph $\tilde{G}_\beta$ (see Fig. 1(a)) and the same graph but with its vertices labeled using their corresponding value of relevance (see Fig. 1(b)). As we can see from this example, the relevance values allow us to reduce the number of vertices to be analyzed, and it also helps us to define an order in which the vertices will be analyzed, for building the covering of $\tilde{G}_\beta$.

The strategy proposed for building a covering of $\tilde{G}_\beta$ is comprised of three steps. First, all vertices are marked as *satellite* and those vertices having relevance greater than zero are added to a *list of candidates L*; isolated vertices are directly included in X since they are degenerated ws-graphs. After, the list L is sorted in descending order according to the relevance of the vertices. Finally, the vertices of L are iteratively visited and each vertex $v \in L$, satisfying at least one of the following conditions, is added to X:

(a) v is not covered yet.
(b) v is already covered but it has at least one adjacent vertex u which is not covered yet. This condition avoids the selection of ws-graphs having all their satellites covered by previously selected ws-graphs.

After all the vertices in L were visited, each selected ws-graph constitutes an initial cluster. In Fig. 2, we show how the above strategy works on the graph of Fig. 1(b). In Fig. 2(a), the vertices

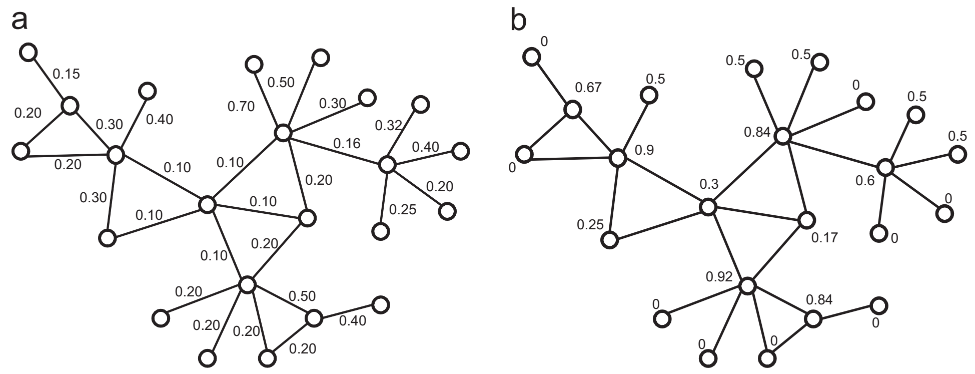

Fig. 1. Illustrating the use of the relevance concept in the process of covering a graph $\tilde{G}_\beta$: (a) a weighted thresholded similarity graph $\tilde{G}_\beta$ and (b) $\tilde{G}_\beta$ using relevance for labeling the vertices.

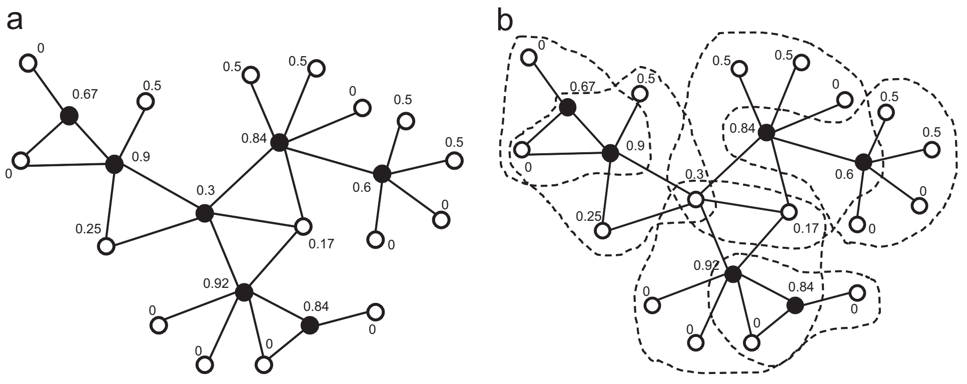

Fig. 2. Illustrating how the strategy proposed for covering $\tilde{G}_\beta$ works: (a) vertices belonging to set X and (b) set of initial clusters.

that were selected and included in the set X are showed filled with black. Finally, in Fig. 2(b) we show the resulting set of initial clusters (ws-graphs).

Once the set X has been built, we analyze it in order to remove those less *useful* ws-graphs. In this context, the usefulness of a ws-graph $G^\star$ will be determined using the number of satellites of $G^\star$ and the number of satellites that $G^\star$ shares with other ws-graphs.

Since for building the initial set of clusters we follow a greedy strategy, it could happen that some of the vertices selected for covering $\tilde{G}_\beta$ are no longer useful for that issue, i.e., they could be removed from X and $\tilde{G}_\beta$ will remain completely covered. For example, if there is a vertex $v \in X$ such that: (i) the ws-graph $G^\star_v$ determined by v shares all their satellites with other selected ws-graphs, and (ii) v itself belongs to at least another selected ws-graph, then we can remove (v from X and the remaining ws-graphs in X would still be a covering of $\tilde{G}_\beta$. Nevertheless, if there is a vertex $u \in X$ such that u meets condition (ii) but it does not meet condition (i), then u cannot be removed from X. Noticed that, even though most of the satellites of $G^\star_u$ would be covered by other ws-graphs, removing u from X will leave uncovered the non-shared satellites of $G^\star_u$. For solving this situation, we will define the usefulness of a ws-graph based on how many satellites it shares.

Let $v \in X$ be a vertex that determines the ws-graph $G^\star_v$ in the covering of $\tilde{G}_\beta$. Additionally, let *v.Shared* be the set of satellites that $G^\star_v$ shares with other selected ws-graphs and *v.Non_shared* be the set of satellites belonging only to $G^\star_v$. In this work, we will understand that $G^\star_v$ is not useful for covering $\tilde{G}_\beta$ iff the following two conditions are met: (1) there is at least another selected ws-graph containing v as a satellite and (2) $|v.Shared| > |v.Non\_shared|$. Otherwise, $G^\star_v$ is useful and it should not be removed from the covering.

Non-useful ws-graphs increase the overlapping of the initial set of clusters; therefore, removing those ws-graphs would help to reduce the number of clusters as well as their overlapping. However, for removing a non-useful ws-graphs we need to add all its non-shared satellites to other clusters. Since a non-useful ws-graph $G^\star_v$ meets that its center v belongs to at least another ws-graph $G^\star$, the non-shared satellites of $G^\star_v$ could be added to the cluster defined by $G^\star$. If there are more than one ws-graph covering the vertex v, then the non-shared satellites will be added to the ws-graph having the greatest number of satellites among the candidates; thus, we allow the creation of clusters with many objects. If there is a tie, then any of the ws-graphs having the greatest number of satellites can be selected. In this work, for simplicity, we select the first of these ws-graphs. When the non-shared satellites of a non-useful ws-graph must be added to another ws-graph $G^\star_v$, we will add those satellites to a list named *v.Linked*; thus, the cluster determined by the ws-graph $G^\star_v$ now will include also the vertices in *v.Linked*. Hereinafter, the vertices added to *v.Linked* will be known as the *linked satellites* of $G^\star_v$.

The strategy proposed for removing non-useful ws-graphs and building the final clustering consists of two steps. First, the set X is sorted in descending order according to the degree of the vertices and each vertex $v \in X$ is marked as *not-analyzed*. In the second step, each vertex $v \in X$ is visited for removing from *v.Adj* all the vertices forming non-useful ws-graphs. For this purpose, each vertex $u \in v.Adj$ is analyzed as follows: If u belongs to X and it is marked as *not-analyzed* (even if it is also marked as *seed*; see below), we check if $G^\star_u$ is a non-useful ws-graph. For doing this, we only need to check if $G^\star_u$ meets the above mentioned condition (2). Noticed that, since u belongs to the ws-graph formed by v, $G^\star_u$ already meets condition (1). After, if $G^\star_u$ is non-useful, then u is removed from X and its non-shared satellites are added to *v.Linked*; otherwise, u is marked as *analyzed*. Once all the vertices of *v.Adj* have been visited, the vertex v is marked as *seed*. Each vertex v marked as *seed* (even those that were removed from X), together with the vertices in *v.Adj* and *v.Linked*, constitutes a cluster in the final clustering.

In Fig. 3, we show how the above filtering strategy works on the set of initial clusters showed in Fig. 2(b). In Fig. 3(a), we show, filled with gray, the vertices determining non-useful ws-graphs and, filled with black, the remaining vertices that form the covering of $\tilde{G}_\beta$. In Fig. 3(b), we show the resulting set of overlapping clusters, obtained after removing the non-useful ws-graphs.

3.3. Updating the clustering when the collection changes

In this section, we explain the strategy that DClustR follows for updating a clustering that has been built using the strategy presented in the previous section. For updating the clustering, DClustR first builds the connected components containing the clusters affected by the changes. Then, DClustR only updates the covering of those components using some previously selected ws-graphs and some new ws-graphs. Following, we will explain which connected components should be analyzed when the collection changes, and also, how to update the covering of those components.

Let $\tilde{G}_\beta = \langle V, \tilde{E}_\beta, S \rangle$ be the weighted thresholded similarity graph, representing a collection O already clustered. Let X be the set of vertices, marked as *seed*, which form the current clustering of O. Since the current clustering was built from the covering of $\tilde{G}_\beta$, for updating the clustering after the changes, first of all, it is important to know how these changes affect the cover of $\tilde{G}_\beta$.

When some vertices are added to and/or removed from G_β, there could happen the following two situations:

(1) Some vertices become uncovered. This situation happens when at least one added vertex is uncovered or when all the vertices $v \in X$ that cover a specific vertex were deleted.
(2) The relevance of some vertices changes and, as a consequence, there appears at least one vertex $u \notin X$ having relevance greater

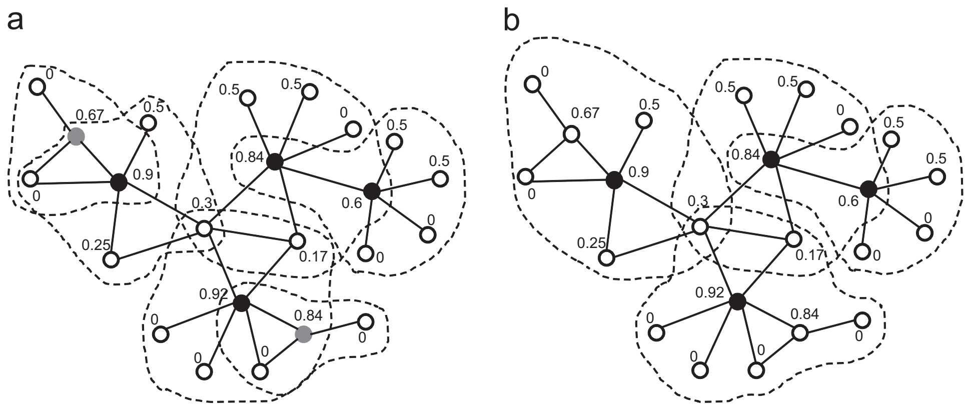

Fig. 3. Illustrating how the strategy proposed for filtering the initial clusters works: (a) vertices determining non-useful ws-graphs and (b) final set of overlapping clusters.

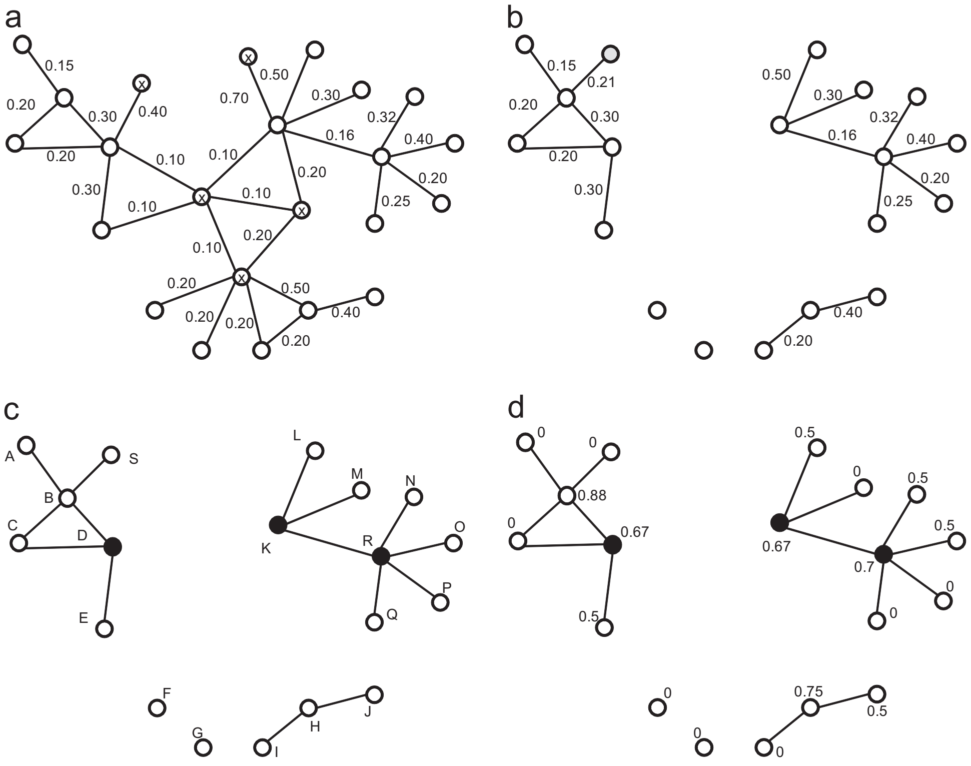

Fig. 4. Illustrating how some changes in the collection affect the current covering of a graph $\tilde{G}_\beta$: (a) $\tilde{G}_\beta$ before the changes; (b) $\tilde{G}_\beta$ after the changes; (c) $\tilde{G}_\beta$ with vertices labeled with letters; (d) $\tilde{G}_\beta$ with vertices labeled with their updated value of relevance.

than at least one vertex $z \in (X \cap u.Adj)$ or greater than at least one vertex $v \in X$, such that v covers at least one vertex in $u.Adj$. Vertices like u could determine ws-graphs with better characteristics (*i.e*, more satellites and less overlapping with other ws-graphs) than those ws-graphs currently belonging to the covering of $\tilde{G}_\beta$.

In Fig 4, we show an example of how the changes in the collection could affect the current covering of the graph $\tilde{G}_\beta$ of Fig. 1 (a). In Fig. 4(a), we show the graph $\tilde{G}_\beta$ before the changes; in this figure, the vertices to be removed are marked with an “x”. In Fig. 4 (b), we show the graph $\tilde{G}_\beta$ after the changes; in this figure, the vertex filled with light gray is a vertex that was added to the graph. Finally, in Fig. 4(c) and (d), we show the updated graph $\tilde{G}_\beta$ with its vertices labeled with letters and with their updated value of relevance, respectively. In Fig. 4(c) and (d), the vertices filled with black correspond with the vertices that belong to set X before the changes.

As it can be seen from Fig. 4(c) and (d), vertices S, F, G, I, H and J are examples of vertices that became uncovered after the changes; that is, these vertices meet the above mentioned situation (1). On the other hand, vertex B is an example of a vertex that meets the above mentioned situation (2); that is, vertex B does not belong to set X and it has, after the changes, a relevance greater than at least one vertex in $(X \cap B.Adj)$ (vertex D).

For updating the current covering when situation (1) happens, we must include in X one vertex u from the set $v.Adj \cup \{v\}$, for each uncovered vertex v; this way, the ws-graph $G^\star_u$ covers the vertex v. On the other hand, for updating the covering when situation (2) happens, a depth analysis is required.

In order to change the relevance of a vertex v, it must change its $v.densityR$ or its $v.compactnessR$. A vertex v could change its relative density if the value of $v.density$ or $|v.Adj|$ changes. Analogously, a vertex v could change its relative compactness if the value of $v.compactness$ or $|v.Adj|$ changes. The value of $|v.Adj|$ changes only if one or more vertices are deleted from/added to $v.Adj$. The value of *v.density* or *v.compactness* could change if *v.Adj* changes or if there is at least one vertex $u \in v.Adj$, such that *u.Adj* changes.

From the previous analysis, we can affirm that a vertex could change its relevance if one or more vertices were added to/removed from its *neighborhood*. The neighborhood of a vertex v consists of the vertices in *v.Adj* plus the adjacent vertices of each vertex in *v.Adj*. Since we are dealing with multiple additions/deletions there could be a lot of overlapping in the neighborhood of the vertices to be analyzed. Therefore, we can say that a vertex could change its relevance if it belongs to a connected component which contains vertices added to $\tilde{G}_\beta$ or vertices that were adjacent to some vertices removed from $\tilde{G}_\beta$. Thus, we can update the covering of $\tilde{G}_\beta$ by updating the covering of the above mentioned connected components. Noticed that, these connected components can be built through a depth first search, starting from the vertices added or from the vertices that were adjacent to those vertices removed from $\tilde{G}_\beta$.

In Fig 5, we show an example of how to build the connected components of $\tilde{G}_\beta$ that were affected by the changes. In Fig. 5(a), the vertices added to $\tilde{G}_\beta$ and the vertices that were adjacent to some vertices removed from $\tilde{G}_\beta$, appear filled with light gray. As it was mentioned above, starting from these vertices, we can build the connected components containing the clusters affected by these changes. In Fig. 5(b), we show how to build each connected component, through a depth first search, starting from vertices E, K, F, G and I, respectively; in Fig. 5(b), the arrows drawn in each component represent the depth first search carried out to build each component. Since vertices F and G are isolated, each one constitutes a connected component by itself.

Let $G' = \langle V', E', S \rangle$ be a connected component whose covering must be updated. Let $X' \subseteq X$ be the set of vertices determining the ws-graphs that cover G'. It is important to mention that, after the changes, some non-useful ws-graphs could be no longer so. Therefore, before updating the covering of G', we will empty the list *v.Linked* for each vertex $v \in X'$; in this way, we will allow the creation of some new ws-graphs. The strategy for updating the covering of G' is comprised of four steps. First of all, we recompute the relevance of all the vertices in V'. After, we build the list L' of candidate vertices (see below). This list contains the vertices that will help us to update the covering of G'; during the construction of L' some vertices would be removed from X'. In the third step, we iteratively select from L' those vertices that, together with the current vertices of X', complete the covering of G'. The strategy used for selecting these new vertices is the same proposed in Section 3.2. After that, in the fourth step, we remove the non-useful ws-graphs from the updated set X'; for this purpose, we use the strategy proposed in Section 3.2. Finally, we only need to explain how to build the list L' from G'.

Let $V_s \subseteq (V' \setminus X')$ be the set of vertices of G' with relevance greater than zero, which do not belong to X'. For computing the candidate list L', both X' and V_s are analyzed. In the processing of V_s, each vertex $v \in V_s$ is visited and the following conditions are verified:

- v is uncovered.
- v has at least one uncovered adjacent vertex.
- There is at least one vertex $u \in v.Adj$, such that all the vertices $z \in X'$, whose ws-graphs $G_z^\star$ cover u, have less relevance than v.

If v meets at least one of the aforementioned conditions, then it is added to L'. Additionally, in cases where the last condition is meet, all vertices u are marked as *active*; this kind of vertices are used during the analysis of X'.

In the processing of X', all the adjacent vertices of each vertex $v \in X'$ are visited. When a vertex $u \in v.Adj$ is visited, if $u.relevance > v.relevance$ then u is added to L' and v is marked as weak. Once all the vertices in *v.Adj* have been visited, if v is marked as *weak* or v has at least one adjacent vertex marked as *active*, then v is removed from X' since it could be replaced by at least another more relevant vertex. Afterwards, if v was removed from X' but its relevance is greater than zero, then v is added to L'.

Once the covering of all the connected components have been updated, each vertex v marked as *seed*, together with the vertices in *v.Adj* and *v.Linked*, constitutes a cluster in the updated clustering. In Fig 6, we show the updated set of overlapping clusters obtained following the above introduced strategy. In this figure, the vertices filled with black represent the vertices that cover each connected component of $\tilde{G}_\beta$.

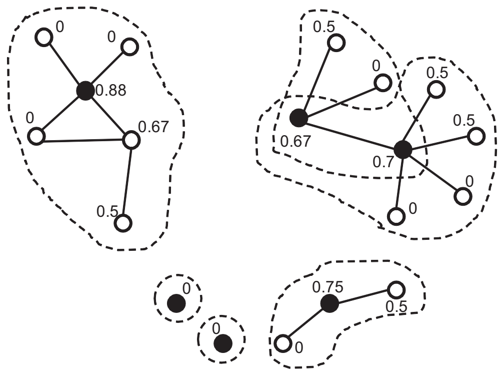

Fig. 6. Updated set of overlapping clusters after applying the strategy proposed for updating the clustering, when the collection of objects represented in the graph of Fig. 1(a) changes as it is showed in Fig. 4(a) and (b).

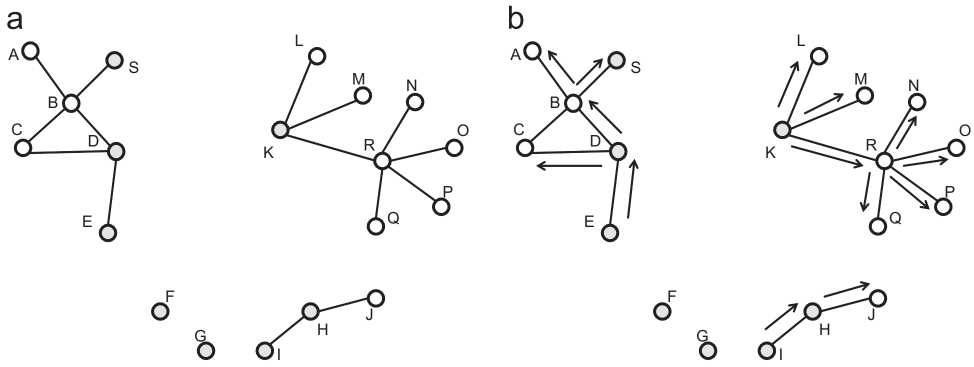

Fig. 5. Illustrating how to build the connected components of $\tilde{G}_\beta$ which were affected by additions and/or deletions of vertices: (a) $\tilde{G}_\beta$ after the changes and (b) building the connected components affected by the changes.

The pseudocode of DClustR is shown in Algorithm 1. For updating a graph after changes in the collection, we use the procedure *UpdGraph*. One of the outputs of this procedure is the set M. This set contains the vertices that were added to $\tilde{G}_\beta$ and/or the vertices that were adjacent to the vertices removed from $\tilde{G}_\beta$. By using these vertices, in the step 5 of Algorithm 1, DClustR builds the connected components affected by the changes. For updating the covering of a connected component G', we use the procedure *UpdCovCompt*. The pseudocodes of *UpdGraph* and *UpdCovCompt* are shown in Algorithms 2 and 3, respectively. In the pseudocode of the procedure *UpdCovCompt*, the conditions (a) and (b) used in step 3, refer to the conditions listed in Section 3.2 for adding vertices to the set X.

Algorithm 1. DClustR algorithm

```
   Input: $O=\{o_1,o_2,\dots,o_n\}$ – collection of objects,
          $\tilde{G}_\beta=\langle V,\tilde{E}_\beta,S\rangle$ – similarity graph,
          $\beta$ – similarity threshold,
          $A$ – objects to be added,
          $R$ – objects to be removed
   Output: $O$ – updated collection of objects,
          $\tilde{G}_\beta$ – updated similarity graph,
          $SC$ – up-dated set of clusters

   // Updating $\tilde{G}_\beta$ due to additions/deletions
1  UpdGraph $(O,\tilde{G}_\beta,\beta,A,R,M)$;

2  "Mark vertices in $\tilde{G}_\beta$ as not-processed";
   // Updating the covering of each connected component affected by the changes
3  foreach vertex $v \in M$ do
4      if $v$ is not marked as processed then
5          "Build the connected component $G'=\langle V',E',S\rangle$ containing $v$";
6          if $|V'|=1$ then "Mark $v$ as seed";
7          else
8              "Recompute relevance of vertices in $G'$";
9              "Build $V_s$ and $X'$";
10             "Emptying $u.Linked$ for each vertex $u \in X'$"; "Mark vertices in $G'$ as processed";
11             "Build candidate list $L'$";
12             UpdCovCompt $(G',L',X')$;
13
   // Returning the updated clustering
14 $SC:=\emptyset$;
15 foreach vertex $v \in V$ do
16     if $v$ is marked as seed then $SC:=SC \cup \{\{v\} \cup v.Adj \cup v.Linked\}$;
```

Algorithm 2. UpdGraph procedure

```
   Input: $O=\{o_1,o_2,\dots,o_n\}$ – collection of objects,
          $\tilde{G}_\beta=\langle V,\tilde{E}_\beta,S\rangle$ – similarity graph,
          $\beta$ – similarity threshold,
          $A$ – objects to be added,
          $R$ – objects to be removed
   Output: $O$ – updated collection of objects,
          $\tilde{G}_\beta$ – updated similarity graph,
          $M$ – Set of vertices used for building the connected
   components affected by the changes

1  $M:=\emptyset$;
   // deletions
2  foreach object $o \in R$ do
3      "Remove from $\tilde{G}_\beta$ the vertex $v$ representing object $o$";
4      $M:=M \cup \{x \mid x \in v.Adj \wedge x$ does not represent any object in $R\}$;
5  $O:=O \setminus R$;
   // additions
6  foreach object $o \in A$ do
7      "Create vertex $v$ for representing object $o$ and add $v$ to $\tilde{G}_\beta$";
8      $M:=M \cup \{v\}$;
9  $O:=O \cup A$;
```

Algorithm 3. UpdCovCompt procedure

```
   Input: $G'=\langle V',E',S\rangle$ – a connected component,
          $L'$ – candidate list,
          $X'$ – vertices that currently cover $G'$
   Output: $G'$ – updated connected component

   // Selecting new vertices for completing the
   covering of $G'$
1  "Sort $L'$ in descending order according to the relevance";
2  foreach vertex $v \in L'$ do
3      if $v$ meets conditions $(a)$ or $(b)$ then $X':=X' \cup \{v\}$;
   // Removing non−useful ws−graphs
```

```
4   "Sort $X'$ in descending order according to the degree";
5   "Mark vertices in $X'$ as not-analyzed";
6   $T:=\emptyset$;
7   foreach vertex $c \in X'$ do
8       foreach vertex $v \in c.Adj$ do
9           if $v \in X'$ and $v$ is marked as not−analyzed then
10              if the ws−graph determined by $v$ is not useful then
11                  $c.Linked:=c.Linked \cup v.Non\_shared$;
12                  $X':=X' \setminus \{v\}$;
13                  $T:=T \cup \{c\}$;
14              else "Mark $v$ as analyzed";
15  "Mark vertices in $T$ as seed";
16  "Mark vertices in $V' \setminus T$ as satellite";
```

As it can be noticed from Algorithm 1, DClustR assumes that there exists a weighted thresholded similarity graph $\tilde{G}_\beta$ representing the current collection. If there is no previous collection and it is the first time that the collection will be clustered, then $\tilde{G}_\beta$ is an empty graph; in this way, DClustR is also able to process a collection starting from scratch. The computational complexity of DClustR algorithm is $O(n^2)$.

It is important to mention that, although DClustR is related to Star [10], ICSD [26] and DCS [27], there are some differences that we would like to highlight. Unlike Star, ICSD and DCS, DClustR defines the concept of *relevance* in function of the relative density and the relative compactness and it builds the clustering based on this property. As we will show in the experiments, the use of the relevance allows DClustR to build clusterings having greater quality than those built by state-of-the-art algorithms, including Star, ICSD and DCS. Additionally, the procedure used by DClustR for improving the initial clustering is totally different from the filtering procedure used by ICSD and DCS; as we will show in our experiments, this procedure allows DClustR to build clusterings with less clusters and less overlapping than those clusterings built by state-of-the-art algorithms. Moreover, unlike Star, the strategy introduced by DClustR for updating the clustering allows to process multiple additions/deletions efficiently.

Finally, we would like to point out that DClustR depends on the data order; that is, DClustR could build different clusterings from the same set of objects, depending the order in which the objects are analyzed. However, as we will show in Section 4, the differences among the quality of those different clusterings are very small.

4. Experimental results

In this section, the results of several experiments testing the performance of the DClustR algorithm are presented.

The experiments were conducted over several overlapping collections and were focused on comparing the algorithms according to: (1) the quality of the clustering, (2) the number of clusters obtained, (3) the overlapping of the clustering, and (4) the time each algorithm spends for processing multiple additions/deletions. In the first three experiments, we contrast the results of DClustR with the results obtained by the algorithms of the state-of-the-art: Star, ISC, SHC, ICSD, DCS and DHS. Additionally, we include in these three experiments a comparison of DClustR against three static overlapping clustering algorithms, which have reported good results: Estar [38], Gstar [39] and ACONS [40]. We would like to highlight that for any of the test collections used in these experiments, the algorithm of Duan et al. [29] was not able to produce a clustering solution after eight hours. This is due to its very high computational complexity, which could get to be exponential, therefore, we did not include Duan's algorithm in these experiments.

In the fourth experiment, for multiple additions, we contrast our results against those of ISC, Star, SHC, ICDS and DCS. For multiple deletions and modifications, we contrast our results against those obtained by Star and DCS, which are the only dynamic overlapping clustering algorithms reported in the literature. Since DHS is $O(n^3)$ and it must build a hierarchy of clustering instead of a single clustering, the comparison between DHS and DClustR (which is $O(n^2)$ and it builds a single clustering) wrt. efficiency would not be fair; therefore, we do not include the DHS algorithm in the fourth experiment.

It is important to mention that, in this section, we compare our proposed algorithm against those algorithms of the state-of-the-art which are more related to our work; that is, we compare DClustR against those algorithms proposed for overlapping clustering which are also able to process changes in the collection. Notice that, a comparison against incremental or dynamic non-overlapping algorithms would not be fair for any algorithm, since they are addressing different problems. Besides, as it was mentioned in Section 2, a direct comparison of our proposed algorithm against fuzzy algorithms is not possible. Thus, we have left as future work the extension of our algorithm to produce fuzzy clusterings as well as the comparison against incremental fuzzy clustering algorithms.

All the algorithms used in the experiments were implemented in C++ and compiled using the g++ compiler. The experiments were performed on a PC with an Intel Core 2 Duo at 1.86 GHz CPU with 2 GB DDR2 RAM, running RedHat Enterprise Linux 5.3.

4.1. Collections used in the experiments

Since we are facing the problem of overlapping clustering, the algorithms should be evaluated over collections with overlapping classes. Therefore, we decided to evaluate the algorithms in the task of document clustering, where it is common that some documents belong to more than one topic.

The document collections used in our experiments were built from five benchmark text collections commonly used in document clustering: AFP, Reuters-21578, TDT2, CISI and CACM. Each one of these benchmarks has a *ground-truth* that is distributed together with the benchmark and which has been manually tagged by experts. The AFP, Reuters-21578 and TDT2 benchmarks can be obtained from http://trec.nist.gov, http://kdd.ics.uci.edu and http://www.nist.gov/speech/tests/tdt.html, respectively. On the other hand, both CISI and CACM can be obtained from ftp://ftp.cs.cornell.edu/pub/smart.

From these benchmarks, 12 document collections were built. The collections AFP, CISI, CACM and TDT were built from the benchmarks AFP, CISI, CACM and TDT2, respectively, using all the news that have been associated with at least one topic in the *ground-truth*. The collections Reu-Te and Reu-Tr were built from the benchmark Reuters-21578, using the news that have been associated with at least one topic in the *ground-truth* and have been tagged as "Test" (Reu-Te) and "Train" (Reu-Tr). The collection Reuter is the union of Reu-Te and Reu-Tr. Finally, five sub-collections of TDT called TDT-1, TDT-2, TDT-3, TDT-4 and TDT-5 were built from TDT. For constructing these last five sub-collections, the news of TDT were randomly arranged into five-folds and, for each sub-collection, three of these five-folds were randomly selected. The characteristics of the document collections used in our experiments are shown in Table 1. In this table, the column "'Overlapping' represents the overlapping of a collection and it is computed as the average number of clusters in which an object is included [28].

Table 1
Overview of collections.

Name	#Documents	#Terms	#Classes	Overlapping
AFP	695	11,785	25	1.023
Reu-Te	3587	15,113	100	1.295
Reu-Tr	7780	21,901	115	1.241
Reuter	11,367	27,083	120	1.258
TDT	16,006	68,019	193	1.188
TDT-1	8602	51,764	176	1.202
TDT-2	7404	44,610	178	1.173
TDT-3	10,258	53,706	174	1.189
TDT-4	10,074	53,036	172	1.182
TDT-5	11,328	55,923	182	1.180
CACM	433	3038	52	1.499
CISI	1162	6976	76	2.680

In our experiments, documents were represented using the vector space model (VSM) [34]. The index terms of the documents represent the lemmas of the words occurring at least once in the collection; these lemmas were extracted from the documents using Tree-tagger.[1] Stop words such as: articles, prepositions and adverbs were removed. The index terms of each document were statistically weighted using term frequency (tf) normalized by the *maximum term frequency*. The *maximum term frequency* is the highest frequency of a term in a given document [41]. It is important to mention, that we also tested other weighting schemes like tf, tf/idf and tf normalized by the logarithm, among others reported in [41]. However, the best results were obtained by the term frequency normalized by the *maximum term frequency* and these are the results that we report in this paper. Finally, the cosine measure was used to compute the similarity between two documents [35].

4.2. Evaluation measures

There are three types of clustering evaluation measures: *external*, *relative* and *internal* measures [1]. From these three kind of measures, the most widely used are the external measures. The external measures evaluate a clustering solution based on how much this clustering resembles a set of classes, commonly known as *ground-truth*, which has been manually tagged by human experts; the more similar the clustering solution is to the *ground-truth*, the better the clustering algorithm is.

Many external evaluation measures have been proposed in the literature, for instance: Purity and Inverse Purity [42], F1-measure [43], Jaccard coefficient [44], Entropy [45], Class Entropy [46] and V-measure [47], among others. These measures are different according to their mathematical foundations, their biases and their limitations. However, none of the external measures reported so far have been developed, at least explicitly, for evaluating overlapping clustering algorithms, i.e., these measures fail at reflecting the fact that, in a perfect overlapping clustering, objects sharing n classes should share n clusters.

In [12], it was proposed a new external measure for evaluating overlapping clusterings. This measure is called FBcubed and it is computed using variations of the Bcubed precision and recall measures [48]. The FBcubed measure meets four constraints which evaluate several desirable characteristics in an overlapping clustering solution. These constraints are intuitive and they express important characteristics that an external evaluation measure should evaluate. Moreover, in [12], the authors showed that none of the most used external evaluation measures satisfies all these four constraints.

[1] http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger

Based on the aforementioned analysis, and in order to conduct a fair comparison among the algorithms, we will use the FBcubed measure for evaluating the quality of the clusterings built by each algorithm. Moreover, to the best of our knowledge, the existing internal an relative evaluation measures are defined for evaluating disjoint clusters; therefore, before applying one of these measures for evaluating overlapping clustering, a deep study is required in order to adapt them for evaluating this type of clusterings. A more detailed explanation about the FBcubed measure, together with a case of study, can be found in [12].

4.3. Quality

In this experiment, we compare the quality of the clusters built by the clustering algorithms for each document collection; for computing the quality of the clusters we use the FBcubed measure [12].

In Table 2, we show the best performances, according to the FBcubed measure, attained by each algorithm over each collection. For obtaining these best performances, we find the parameter values for which each algorithm obtained its highest FBcubed value over each collection. For this purpose, we proved different values of β in [0.05; 0.50], with an increment of 0.01; that is, we used $\beta = 0.05$, 0.06, 0.07 and so on. For those algorithms that depend on data order: DClustR, Star, Gstar, ACONS, ICSD, DCS and SHC, we repeated the experiment 20 times, for each parameter value, varying the order of the documents and, after that, we computed for each parameter value, the average FBcubed obtained by each algorithm; for these algorithms, the best performance corresponds to the highest average FBcubed. For the DHS algorithm, we computed the FBcubed of the clustering at each level of the hierarchies built for each parameter value; for this algorithm, the best performance corresponds with the FBcubed value of the level having the highest FBcubed value.

During this experiment, we realized that even when Star, SHC, GStar, ACONS, ICSD, DCS and DClustR depend on data order, the standard deviation of their FBcubed values was less than 0.01. Besides, in this experiment we realized that, for all the collections, the top level of the hierarchies, built by DHS, obtained the highest FBcubed value. In this experiment, we also realized that for values of β greater than 0.50 or smaller than 0.10, the quality of the algorithms Star, ISC, Estar, Gstar, ACONS, ICSD, SHC, DCS and DClustR decreases; besides, we observed that for values of β greater than 0.35 or smaller than 0.05, the quality of the DHS algorithm decreases. For the above reasons, we do not use values of β out of [0.05; 0.50], for any of the tested algorithms.

As it can be seen from Table 2, DClustR builds in almost all collections, higher quality clusterings than all the other algorithms, according to the FBcubed measure. For summarizing the above results, we employed an experimental methodology similar to that used in [49,28]. Table 3 shows the statistical significance matrix for the FBcubed values obtained by each algorithm. In this matrix, the symbols "$>>$" ("$<<$") indicates that the FBcubed value obtained by the algorithm of the row are significantly better (worse) than the value obtained by the algorithm of the column; the symbol "$>$" ("$<$") indicates that the relation is not significant. For testing the statistical significance we used the Mann–Whitney test, with a 95% of confidence. A detailed explanation about this test, as well as an implementation, can be found at http://faculty.vassar.edu/lowry/webtext.html.

As it can be seen from Table 3, excepting the DHS algorithm, DClustR significantly wins to the other algorithms used in the comparison, in terms of the quality of the clusters. Even when in almost all collections (10 from 12 collections) DClustR attains higher FBcubed values than those attained by DHS, the differences are not significant according to the Mann–Whitney test. However, it is important to remember that our proposed algorithm has a lower computational complexity than DHS; thus, DClustR will be

Table 2
Best performances of each algorithm over each document collection. The highest values per collection appear bold-faced.

Collection	Star	ISC	Estar	Gstar	ACONS	ICSD	SHC	DCS	DHS	DClustR
AFP	0.69	0.20	0.63	0.63	0.62	0.61	0.27	0.61	**0.80**	0.77
Reu-Te	0.45	0.05	0.39	0.40	0.40	0.39	0.20	0.39	0.49	**0.51**
Reu-Tr	0.42	0.03	0.36	0.36	0.36	0.36	0.19	0.36	**0.44**	0.43
Reuter	0.42	0.02	0.34	0.35	0.36	0.35	0.19	0.35	0.42	**0.43**
TDT	0.43	0.06	0.37	0.35	0.34	0.35	0.15	0.35	0.45	**0.48**
TDT-1	0.45	0.09	0.39	0.38	0.38	0.38	0.16	0.38	0.45	**0.48**
TDT-2	0.47	0.10	0.40	0.40	0.39	0.40	0.17	0.40	0.47	**0.52**
TDT-3	0.46	0.07	0.40	0.40	0.39	0.39	0.17	0.39	0.48	**0.51**
TDT-4	0.46	0.07	0.40	0.39	0.39	0.39	0.17	0.39	0.48	**0.50**
TDT-5	0.46	0.07	0.39	0.37	0.37	0.37	0.16	0.37	0.48	**0.50**
CACM	0.31	0.18	0.32	0.31	0.32	0.32	0.15	0.32	0.29	**0.33**
CISI	0.30	0.05	0.29	0.29	0.29	0.29	0.21	0.29	0.29	**0.32**

Table 3
Statistical significance matrix for FBcubed values.

Algorithm	Star	ISC	Estar	Gstar	ACONS	ICSD	SHC	DCS	DHS	DClustR
Star	–	> >	> >	> >	> >	> >	> >	> >	> >	< <
ISC	< <	–	< <	< <	< <	< <	< <	< <	< <	< <
Estar	< <	> >	–	>	>	>	> >	>	>	< <
Gstar	< <	> >	<	–	>	>	> >	>	>	< <
ACONS	< <	> >	<	<	–	>	> >	>	>	< <
ICSD	< <	> >	<	<	<	–	> >	>	>	< <
SHC	< <	> >	< <	< <	< <	< <	–	< <	< <	< <
DCS	< <	> >	<	<	<	<	> >	–	< <	< <
DHS	>	> >	> >	> >	> >	> >	> >	> >	–	<
DClustR	> >	> >	> >	> >	> >	> >	> >	> >	>	–

Table 4
Number of clusters built by each algorithm for each collection. The smallest values per collection appear bold-faced.

Collection	Star	ISC	Estar	Gstar	ACONS	ICSD	SHC	DCS	DHS	DClustR
AFP	123	334	98	90	129	104	85	104	**36**	52
Reu-Te	507	1785	600	711	798	621	273	621	**43**	102
Reu-Tr	471	3936	904	849	857	853	561	853	**16**	166
Reuter	583	5726	659	1532	1420	1183	815	1183	**23**	211
TDT	2019	8250	1854	1653	1663	1657	1203	1657	**84**	769
TDT-1	1184	4425	1207	1075	1077	1078	643	1078	**62**	377
TDT-2	970	3743	1074	948	954	950	579	950	**84**	388
TDT-3	1338	5253	1355	1187	1196	1190	758	1190	**67**	594
TDT-4	1104	5154	1303	1158	1163	1160	731	1160	**58**	434
TDT-5	1425	5816	1451	1291	1295	1293	837	1293	**63**	614
CACM	124	228	122	129	129	152	**29**	152	**29**	102
CISI	134	654	195	159	213	209	100	209	**7**	52

able to process a collection in less time than DHS. Moreover, unlike the DHS algorithm, DClustR does not impose any constraints over the objects of the collection neither it leaves uncovered objects, i. e., objects that do not belong to any cluster.

4.4. Number of clusters

In this experiment, we compare the algorithms according to the number of clusters they build for each document collection, when they attain their best performance according to clustering quality (see Table 2). Table 4 shows the number of clusters built by each algorithm for each document collection.

As it can be noticed from Table 4, DHS builds less clusters than the other tested algorithms. This behavior is expected taking into account that, as it was mentioned in the previous experiment, the highest FBcubed values of the hierarchies built by DHS are attained at the top level. It is important to remember that DHS is an agglomerative hierarchical algorithm and these kind of algorithms build a hierarchy of clusterings by iteratively merging the most similar clusters; therefore, it is usual that the top level of these hierarchies has few clusters. However, it is important to remember that DHS attains lower quality values than our proposed algorithm. Besides, DHS has a higher computational complexity than DClustR and it has several limitations that our proposed algorithm does not have.

We considered important to highlight that, excluding the DHS which addresses a different problem to the rest of the tested algorithms, our proposed algorithm builds, in almost all collections, less clusters than the other algorithms. Although for collection CACM SHC builds less clusters that DClustR, the clusterings built by SHC have poor quality wrt. the quality of the clusterings built by DClustR (see Table 2). These results mean that, in almost all collections, the best clustering built by DClustR has less clusters than the best clustering built by the other non-hierarchical algorithms; in other words, the other non-hierarchical algorithms attain their best quality results at the expense of increasing the number of clusters. Besides, another possible reason for this behavior is that all other non-hierarchical algorithms do not detect

Table 5
Overlapping of the clustering built by each algorithm for each document collection. The lowest values per collection appear bold-faced.

Collection	Star	ISC	Estar	Gstar	ACONS	ICSD	SHC	DCS	DHS	DClustR
AFP	1.71	1.65	2.52	2.31	2.48	2.53	2.43	2.53	**1.01**	1.18
Reu-Te	3.41	1.79	7.40	6.73	6.66	7.64	13.13	7.64	**1.01**	1.40
Reu-Tr	5.54	1.84	12.14	13.08	12.65	13.32	29.33	13.32	**1.01**	1.56
Reuter	5.46	1.82	15.92	15.47	15.47	19.25	47.55	19.25	**1.01**	1.53
TDT	4.81	1.88	59.41	69.43	66.38	70.97	80.74	70.97	**1.07**	1.50
TDT-1	3.38	1.84	44.22	49.08	46.40	49.63	47.91	49.63	**1.06**	1.43
TDT-2	3.38	1.80	35.11	37.85	37.46	37.85	39.82	37.85	**1.15**	1.39
TDT-3	3.81	1.84	42.40	46.08	44.83	46.22	53.79	46.22	**1.09**	1.53
TDT-4	4.08	1.83	43.34	47.59	46.24	48.72	56.04	48.72	**1.04**	1.45
TDT-5	3.98	1.88	44.03	51.46	48.44	52.23	59.79	52.23	**1.12**	1.45
CACM	2.31	1.82	3.46	3.20	3.19	2.72	1.99	2.72	**1.04**	1.26
CISI	4.12	2.12	7.85	7.49	7.77	7.54	6.98	7.54	**1.01**	1.58

big clusters; instead, these algorithms divide those big clusters into many small clusters. Thus, we can affirm that DClustR builds clusterings that could be easier to analyze than those clusterings built by the other non-hierarchical algorithms of the state-of-the-art.

4.5. Overlapping

In this experiment, we compare the algorithms according to the overlapping they produce for each document collection, when they attain their best performance according to clustering quality (see Table 2). The overlapping of a clustering is computed as the average number of clusters in which an object is included [28]. In Table 5, we show the overlapping of the clusterings built by each algorithm for each document collection.

From Table 5, it can be seen that DHS builds clusterings with less overlapping than the clusterings built by the other algorithms and that DClustR is the second best according to this aspect. Nevertheless, as it was mentioned in the previous two sections, it is important to remember that our proposed algorithm has lower computational complexity than DHS. Besides, DClustR does not impose any constraint over the objects neither it leaves uncovered objects, as DHS does. Notice that, since the overlapping point of view, DClustR is the one that performs the best among the non-hierarchical clustering algorithms; this way, DClustR allows overlapping among the clusters but it controls the overlapping in order to avoid building clusters with a so high overlapping that they could be interpreted as the same cluster.

4.6. Behavior for multiple additions/deletions

In these experiments we compare the time spent by each algorithm for processing multiple additions and/or deletions over the largest collection used in the previous experiments, i.e., TDT. In the experiments with multiple additions, we compare the time spent by Star, ICSD, ISC, SHC, DCS and DClustR. In the experiments with multiple deletions and modifications, we compare the time spent by Star, DCS and DClustR, because only these three algorithms are able to process deletions and modifications. As it was mentioned before, since DHS is $O(n^3)$ and it must build a hierarchy of clustering instead of a single clustering, the comparison between DHS and the other non-hierarchical algorithms (which also have a lower computational complexity than DHS) would not be fair; therefore, we do not include DHS in this experiment.

Fig. 7 shows the behavior of the different algorithms for multiple additions. In this figure, each curve represents the average time spent by each algorithm for clustering sub-collections of size 2000, 4000, 6000 and so on; this average time was computed over 20 executions of the algorithms over the TDT collection, randomly varying the order of the documents.

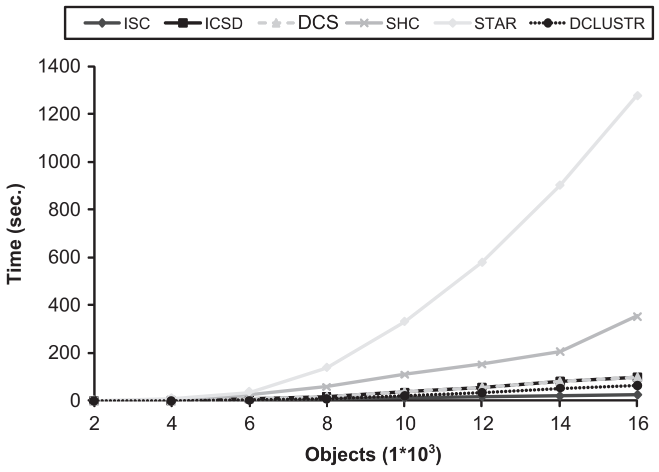

Fig. 7. Behavior of each algorithm under multiple additions over TDT.

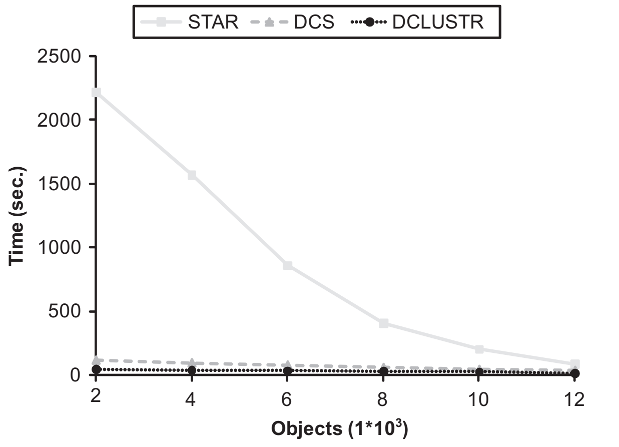

Fig. 8. Behavior of Star, DCS and DClustR under multiple deletions over TDT.

Fig. 8 shows the behavior of the different algorithms for multiple deletions. In this figure, each curve represents the average time spent by each algorithm for updating the clustering every time 2000 documents were randomly removed from the collection; this average time was computed over 20 executions of the algorithms over the TDT collection.

Fig. 9 shows the behavior of the different algorithms for multiple modifications. As it was mentioned before, Star, DCS and DClustR algorithms process a modification as a deletion followed by an addition. In this figure, each curve represents the average time spent by each algorithm for updating the clustering

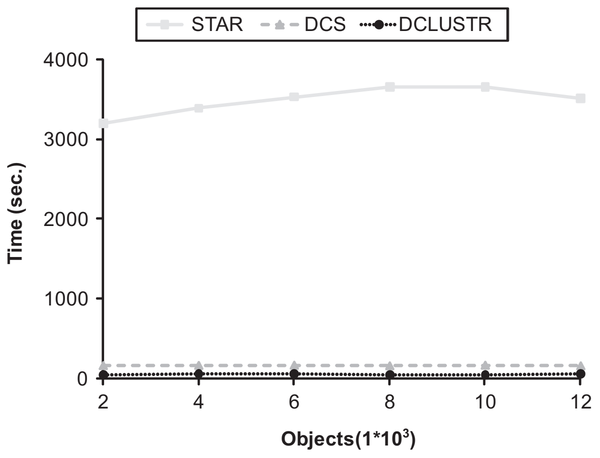

Fig. 9. Behavior of Star, DCS and DClustR under multiple modifications over TDT.

every time 2000 documents were randomly deleted and added but randomly increasing or decreasing the weight of some terms belonging to the documents by a factor of 0.05 (i.e., modified); this average time was computed over twenty executions of the algorithms over the TDT collection.

As it can be seen from Fig. 7, DClustR has a better performance than Star, SHC, ICSD and DCS, for processing multiple additions over the TDT collection. Although ISC has a slightly better performance than our proposed algorithm, it is important to mention that DClustR builds clusters with higher quality than those built by ISC (see Section 4.3) and that ISC cannot process deletions. Additionally, as it can be seen from Figs. 8 and 9, our proposed algorithm clearly overcomes Star and DCS in the processing of multiple deletions and modifications over TDT. We conducted other experiments varying the number of objects that are added, removed or modified and we observed the same behavior.

As it was mentioned in Section 4, the algorithm proposed by Duan et al. [29] was not able to produce a clustering solution for any of the test collections; therefore, it was not included in the previous experiments with TDT collection. However, we will compare it against our algorithm, using the datasets that Duan et al. employed in [29] for evaluating their algorithm: ENRON and DBLP. The ENRON dataset contains the e-mail communication data[2] of ENRON company from 27 September 1999 to 5 March 2000. We took from this dataset a sample comprised of 6097 different e-mail addresses (nodes) and 11,741 messages (edges), distributed over 23 time-slices. The DBLP dataset consists of papers published in five proceedings (KDD, ICDM, CIKM, WWW and SIGIR) from 2006 to 2009 extracted form DBLP.[3] We took from this dataset a sample comprised of 1174 authors (nodes) and 3560 co-authorships (edges), distributed over four time-slices.

In Fig. 10(a) and (b), we show the behavior of our proposed algorithm and the algorithm of Duan et al. over the DBLP and ENRON datasets, respectively. In these figures, the curves labeled with INC-2, INC-3 and INC-4 correspond with the behavior of the algorithm of Duan et al. for $k=2$, 3 and 4, respectively; these are the parameters used in the experiments presented in [29].

As it can be noticed from Fig. 10, the proposed algorithm clearly outperforms the algorithm of Duan et al. in both ENRON and DBLP datasets. Additionally, we would like to mention that since both ENRON and DBLP datasets do not have a *ground-truth*, we were not able to compare DClustR and Duan's algorithm, according to the quality of the clusterings produced over these two datasets.

However, in order to compare in some way the quality of the clusterings obtained by DClustR and the Duan's algorithm, we built one sub-collection for each collection that appears in Table 1, by randomly selecting only 100 of its documents. For values greater than 100, the Duan's algorithm did not get results after 6 h. In fact, in this experiment we only show the results over the AFP, CACM and CISI sub-collections because for the other sub-collections the Duan's algorithm did not get a result after six hours. In Table 6, we show the best performances, according to the FBcubed measure, attained by both algorithms over the AFP, CACM and CISI sub-collections. In this table, the columns labeled INC-2, INC-3 and INC-4 correspond with the performance of the Duan's algorithm for $k=2$, 3 and 4, respectively.

As it can be seen from Table 6, our proposed algorithm builds better quality clusterings than the Duan's algorithm in terms of the FBcubed measure, in all the tested sub-collections.

5. Conclusions

In this paper, we introduced DClustR, a new dynamic clustering algorithm for building overlapping clusters. DClustR introduces a new strategy for building an overlapping clustering and a new strategy for efficiently updating the clustering when the collection changes due to multiple additions and/or deletions. Additionally, like the other algorithms analyzed in this work, our algorithm does not depend on a specific similarity measure.

The proposed algorithm was compared against several overlapping clustering algorithms reported in the literature, using several standard overlapping collections. The experimental evaluation was focused on comparing the algorithms according to the quality of the clusters, the number of clusters, the overlapping of the clusters and the time spent for processing multiple additions and/or deletions. From these experiments, we can conclude that among all the overlapping clustering algorithms used in the experiments, our proposed algorithm is the best, according to the FBcubed evaluation measure. Additionally, excepting the DHS algorithm, DClustR builds clusterings with less clusters and less overlapping than those clusterings built by the other tested algorithms. It is important to highlight that, even though DHS builds clusterings with less clusters and less overlapping than DClustR, our proposed algorithm neither impose any constraint over the objects of the collection nor it leaves uncovered objects (i.e., objects that do not belong to any cluster), as DHS does. Moreover, DClustR has lower computational complexity than DHS; thus, DClustR will be able to process a collection in less time than DHS.

The experiments also showed that the strategy proposed in DClustR, for processing multiple additions, deletions as well as modifications of objects, is clearly faster than the one used by the Star algorithm which updates the clusters processing the changes one by one. Additionally, DClustR outperforms the incremental algorithms SHC and ICSD in the processing of multiple additions. Finally, although ISC is faster than DClustR for processing multiple additions, our proposed algorithm outperforms ISC regarding quality, number of clusters and overlapping of the obtained clusters. Besides, ISC cannot process deletions of objects. Thus, DClustR has a better trade off between quality and efficiency than the ISC algorithm. Besides, from the efficiency experiments, we observed that DClustR outperforms the algorithm proposed by Duan et al.

Based on all the above, we can conclude that DClustR is a better option for overlapping clustering in a dynamic context, than the algorithms previously reported in the literature.

As future work, we will explore the use of DClustR on hierarchical clustering problems in order to increase its

[2] http://www.cs.cmu.edu/enron/
[3] http://dblpvis.uni-trier.de/

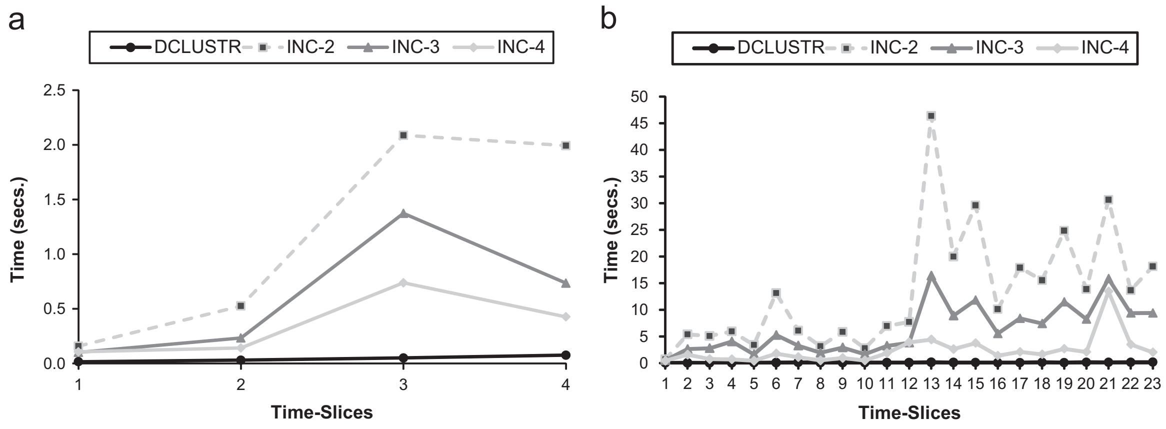

Fig. 10. Behavior of DClustR and the algorithm of Duan et al. for $k=2$, 3 and 4, over DBLP and ENRON datasets. (a) DBLP and (b) ENRON.

Table 6
Best performances of DClustR and Duan's algorithms over the AFP, CACM and CISI sub-collections. The highest values per sub-collection appear bold-faced.

Sub-collection	INC-2	INC-3	INC-4	DClustR
AFP	0.73	0.71	0.70	**0.78**
CACM	0.44	0.42	0.39	**0.47**
CISI	0.41	0.43	0.40	**0.46**

application scope. Besides, we will study other alternatives for combining the relative density and the relative compactness.

Conflict of interest statement

None declared.

Acknowledgments

We thank to the National Council on Science and Technology of Mexico (CONACyT), for its support to this research through the project Grants CB2008-106443 and CB2008-106366.

References

[1] D. Pfitzner, R. Leibbrandt, D. Powers, Characterization and evaluation of similarity measures for pairs of clusterings, Knowledge and Information Systems 19 (3) (2009) 361–394.
[2] L. Hsiu-Hsia, S. San-Ging, L. Yueh-Huang, Y. Shyr-Shen, Bone age cluster assessment and feature clustering analysis based on phalangeal image rough segmentation, Pattern Recognition 45 (1) (2012) 322–332, http://dx.doi.org/10.1016/j.patcog.2011.06.003.
[3] V. Michel, A. Gramfort, G. Varoquaux, E. Eger, C. Keribin, B. Thirion, A supervised clustering approach for fMRI-based inference of brain states, Pattern Recognition 45 (6) (2012) 2041–2049, http://dx.doi.org/10.1016/j.patcog.2011.04.006.
[4] D. Xiang-Ying, C. Qing-Cai, W. Xiao-Long, X. Jun, Online topic detection and tracking of financial news based on hierarchical clustering, in: Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC 2010), 2010, pp. 3358–3361.
[5] A. Ducournau, A. Bretto, S. Rital, B. Laget, A reductive approach to hypergraph clustering: an application to image segmentation, Pattern Recognition 45 (7) (2012) 2788–2803, http://dx.doi.org/10.1016/j.patcog.2012.01.005.
[6] A. Narasimhamurthy, D. Greene, N. Hurley, P. Cunningham, Partitioning large networks without breaking communities, Knowledge and Information Systems 25 (2) (2010) 345–369.
[7] I. Maraziotis, A semi-supervised fuzzy clustering algorithm applied to gene expression data, Pattern Recognition 45 (1) (2012) 637–648, http://dx.doi.org/10.1016/j.patcog.2011.05.007.
[8] L. Chang, M. Duarte, L. Sucar, E. Morales, Object class recognition using SIFT and Bayesian networks, in: Proceedings of the Mexican International Conference on Artificial Intelligence (MICAI 2010), Lecture Notes in Artificial Intelligence, vol. 6438, 2010, pp. 56–66.
[9] R. Abella-Pérez, J. Medina-Pagola, An incremental text segmentation by clustering cohesion, in: Proceedings of the International Workshop on Handling Concept Drift in Adaptive Information Systems: Importance, Challenges and Solutions (HaCDAIS 2010), 2010, pp. 65–72.
[10] J. Aslam, K. Pelekhov, D. Rus, Static and dynamic information organization with star clusters, in: Proceedings of the 7th International Conference on Information and Knowledge Management, 1998, pp. 208–217.
[11] A. Pons-Porrata, J. Ruiz-Shulcloper, R. Berlanga-Llavorí, Y. Santiesteban-Alganza, Un algoritmo incremental para la obtención de cubrimientos con datos mezclados, Reconocimiento de Patrones. Avances y Perspectivas. Research on Computing Science, CIARP2002, 2002, pp. 405–416.
[12] E. Amigó, J. Gonzalo, J. Artiles, F. Verdejo, A comparison of extrinsic clustering evaluation metrics based on formal constraints, Information Retrieval 12 (2009) 461–486.
[13] T. Havens, J. Bezdek, C. Leckie, L. Hall, M. Palaniswami, Fuzzy c-means for very large data, IEEE Transactions on Fuzzy Systems PP (99) (2012) 1–1. http://dx.doi.org/10.1109/TFUZZ.2012.2201485.
[14] S. Okada, T. Nishida, Online incremental clustering with distance metric learning for high dimensional data, in: Proceedings of the 2011 International Joint Conference on Neural Networks (IJCNN), 2011, pp. 2047–2054.
[15] H. Ning, W. Xu, Y. Chi, Y. Gong, T. Huang, Incremental spectral clustering by efficiently updating the eigen-system, Pattern Recognition 43 (2010) 113–127.
[16] L. Tao, S. Anand, HIREL: an incremental clustering algorithm for relational datasets, in: Proceedings of the 8th IEEE International Conference on Data Mining, 2008, pp. 887–892.
[17] J. Correa-Morris, D. Espinosa-Isidrón, D. Álvarez Nadiozhin, An incremental nested partition method for data clustering, Pattern Recognition 43 (2010) 2439–2455.
[18] S. Guha, N. Mishra, R. Motwani, L. O'Callaghan, Clustering data streams, in: Proceedings of the IEEE FOCS Conference, 2000, pp. 359–366.
[19] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, R. Motwani, Streaming-data algorithms for high-quality clustering, in: Proceedings of the 18th International Conference on Data Engineering (ICDE '02), 2002, pp. 685–709.
[20] C. Aggarwal, J. Han, J. Wang, P. Yu, A framework for clustering evolving data streams, in: Proceedings of the 29th International Conference on Very Large Data Bases, VLDB'2003, VLDB Endowment, 2003, pp. 81–92, URL ⟨http://portal.acm.org/citation.cfm?id=1315451.1315460⟩.
[21] D. Chakrabarti, R. Kumar, A. Tomkins, Evolutionary clustering, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006, pp. 554–560.
[22] Y. Chi, X. Song, D. Zhou, K. Hino, B. Tseng, Evolutionary spectral clustering by incorporating temporal smoothness, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'07, ACM, New York, NY, USA, 2007, pp. 153–162, http://dx.doi.org/http://doi.acm.org/10.1145/1281192.1281212.
[23] A. Ahmed, E. Xing, Dynamic non-parametric mixture models and the recurrent chinese restaurant process: with applications to evolutionary clustering, in: Proceedings of the Eighth SIAM International Conference on Data Mining (SDM2008), 2008.
[24] O. Zamir, O. Etziony, Web document clustering: a feasibility demonstration, in: Proceedings of the 21st Annual International ACM SIGIR Conference, 1998, pp. 46–54.
[25] K. Hammouda, M. Kamel, Efficient phrase-based document indexing for web document clustering, IEEE Transactions on Knowledge and Data Engineering 16 (10) (2004) 1279–1296.
[26] A. Pérez-Suárez, J. Martínez-Trinidad, J. Carrasco-Ochoa, J. Medina-Pagola, A new incremental algorithm for overlapped clustering, in: Proceedings of the 14th Iberoamerican Congress on Pattern Recognition (CIARP2009), Lecture Notes in Computer Sciences, vol. 5856, 2009, pp. 497–504.

[27] A. Pérez-Suárez, J. Martínez-Trinidad, J. Carrasco-Ochoa, J. Medina-Pagola, A dynamic clustering algorithm for building overlapping clusters, Intelligent Data Analysis 16 (2) (2012) 211–232.
[28] R. Gil-García, A. Pons-Porrata, Dynamic hierarchical algorithms for document clustering, Pattern Recognition Letters 31 (6) (2010) 469–477.
[29] D. Duan, Y. Li, R. Li, Z. Lu, Incremental *K*-clique clustering in dynamic social networks, Artificial Intelligence Review 38 (2012) 129–147.
[30] S. Gregory, An algorithm to find overlapping community structure in networks, in: Proceedings of the PKDD 2007, Lecture Notes in Artificial Intelligence, vol. 4702, 2007, pp. 91–102.
[31] S. Gregory, A fast algorithm to find overlapping communities in networks, in: Proceedings of the 12th ECML KDD, Lecture Notes in Artificial Intelligence, vol. 5212, 2008, pp. 408–423.
[32] D. Gusfield, Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997. (Chapter 6).
[33] R. Sibson, An optimally efficient algorithm for the single link cluster method, Computer Journal 16 (1973) 30–34.
[34] G. Salton, A. Wong, C. Yang, A vector space model for automatic indexing, Communications of the ACM 18 (11) (1975) 613–620 http://dx.doi.org/doi: http://doi.acm.org/10.1145/361219.361220.
[35] M. Berry, Survey of Text Mining, Clustering, Classification and Retrieval, Springer-Verlag, 2004.
[36] G. Palla, I. Derényi, I. Farkas, T. Vicsek, Uncovering the overlapping community structure of complex networks in nature and society, Nature 435 (7043) (2005) 814–824.
[37] T. Jo, M. Lee, The evaluation measure of text clustering for the variable number of clusters, in: Proceedings of the 4th International Symposium on Neural Networks: Part II – Advances in Neural Networks, 2007, pp. 871–879.
[38] R. Gil-García, J. Badía-Contelles, A. Pons-Porrata, Extended star clustering algorithm, in: Proceedings of the 8th Iberoamerican Congress on Pattern Recognition (CIARP2003), Lecture Notes in Computer Science, vol. 2905, 2003, pp. 480–487.
[39] A. Pérez-Suárez, J. Medina-Pagola, A clustering algorithm based on generalized stars, in: Proceedings of the 5th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM 2007), Lecture Notes in Artificial Intelligence, vol. 4571, 2007, pp. 248–262.
[40] A. Gago-Alonso, A. Pérez-Suárez, J. Medina-Pagola, ACONS: a new algorithm for clustering documents, in: Proceedings of the 12th Iberoamerican Congress on Pattern Recognition (CIARP2007), Lecture Notes in Computer Science, vol. 4756, 2007, pp. 664–673.
[41] E. Greengrass, Information Retrieval: A Survey, Technical Report TR-R52-008-001, 2001.
[42] Y. Zhao, G. Karypis, Criterion Functions for Document Clustering: Experiments and Analysis, Technical Report 01-40, Department of Computer Science, University of Minnesota, Minneapolis, MN, 2001.
[43] B. Larsen, C. Aone, Fast and effective text mining using linear-time document clustering, Knowledge Discovery and Data Mining (1999) 16–22.
[44] M. Halkidi, Y. Batistakis, M. Vazirgiannis, On clustering validation techniques, Journal of Intelligent Information Systems 17 (2–3) (2001) 107–145.
[45] M. Steinbach, G. Karypis, V. Kumar, A comparison of document clustering techniques, in: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2000), 2000.
[46] J. Bakus, M. Hussin, M. Kamel, A SOM-based document clustering using phrases, in: Proceedings of the 9th International Conference on Neural Information Processing (ICONIP'02), 2002, pp. 2212–2216.
[47] A. Rosenberg, J. Hirschberg, V-measure: a conditional entropy-based external cluster evaluation measure, in: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), 2007, pp. 410–420.
[48] A. Bagga, B. Baldwin, Entity-based cross-document coreferencing using the vector space model, in: Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics, 1998, pp. 79–85.
[49] Y. Zhao, G. Karypis, U. Fayyad, Hierarchical clustering algorithms for document datasets, Data Mining and Knowledge Discovery 10 (2) (2005) 141–168.

Airel Pérez-Suárez was born in 1979. He received his B.S. in Computer Science from the Havana University, Cuba, in 2002. He received his M.Sc. and Ph.D. degrees in Computational Sciences from the National Institute of Astrophysics, Optics and Electronics (INAOE), Mexico, in 2008 and 2011, respectively. He is currently an Aggregate Researcher of the Data Mining Department at the Advanced Technologies Application Centre (CENATAV), Cuba.

José Fco. Martínez-Trinidad received his B.S. and M.Sc. degrees in Computer Science from the Autonomous University of Puebla, Mexico, in 1995 and 1997, respectively, and his Ph.D. degree in the National Polytechnic Institute, Mexico, in 2000. Professor Martinez-Trinidad edited/authored seven books and over one hundred papers, on subjects related to pattern recognition.

Jesús A. Carrasco-Ochoa received his Ph.D. degree in Computer Science from the Center for Computing Research of the National Polytechnic Institute (CIC-IPN), Mexico, in 2001. He works as full time researcher at the National Institute for Astrophysics, Optics and Electronics of Mexico. He has published more than 100 papers on topics related to Pattern Recognition and Data Mining, and co-edited seven books. His current research interests include Logical Combinatorial Pattern Recognition, Data Mining, Testor Theory, Feature and Prototype Selection, Text Analysis, Fast Nearest Neighbor Classifiers and Clustering.

José E. Medina-Pagola received a B.Sc. in Cybernetic Mathematics from the Havana University in 1977 and his Ph.D. from the Higher Polytechnic Institute “José A. Echeverría” (ISPJAE) in 1996. His research interests include but not restricted to knowledge discovery and data mining, association rules, clustering, computational linguistic, information retrieval and text mining. He is currently a Senior Researcher and Research Deputy Director of the Advanced Technologies Application Centre (CENATAV), Cuba.