A dynamic clustering algorithm for building overlapping clusters

Article *in* Intelligent Data Analysis · January 2012

DOI: 10.3233/IDA-2012-0520

CITATIONS
7

READS
113

4 authors, including:

Airel Perez-Suarez
Centro de Aplicaciones de Tecnologias de Avanzada
24 PUBLICATIONS **92** CITATIONS
SEE PROFILE

José Francisco Martínez-Trinidad
Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE)
230 PUBLICATIONS **1,258** CITATIONS
SEE PROFILE

José E. Medina Pagola
University of Information Sciences
62 PUBLICATIONS **260** CITATIONS
SEE PROFILE

Intelligent Data Analysis 16 (2012) 211–232
DOI 10.3233/IDA-2012-0520
IOS Press

A dynamic clustering algorithm for building overlapping clusters

Airel Pérez-Suárez[a,b,*], José Fco. Martĭnez-Trinidad[a], Jesús A. Carrasco-Ochoa[a] and José E. Medina-Pagola[b]
[a]*Computer Science Department, National Institute of Astrophysics, Puebla, Mexico*
[b]*Advanced Technologies Application Center, Siboney, Playa, Havana, Cuba*

Abstract. Clustering is a Data Mining technique which has been widely used in many practical applications. In some of these applications like, medical diagnosis, categorization of digital libraries, topic detection and others, the objects could belong to more than one cluster. However, most of the clustering algorithms generate disjoint clusters. Moreover, processing additions, deletions and modifications of objects in the clustering built so far, without having to rebuild the clustering from the beginning is an issue that has been little studied. In this paper, we introduce DCS, a clustering algorithm which includes a new graph-cover strategy for building a set of clusters that could overlap, and a strategy for dynamically updating the clustering, managing multiple additions and/or deletions of objects. The experimental evaluation conducted over different collections demonstrates the good performance of the proposed algorithm.

Keywords: Data mining, overlapping clustering, graph-based algorithms

1. Introduction

Nowadays, many applications using data mining techniques process datasets that could change over time [38]. A data mining technique, which has been widely used in many applications, is clustering [29]. Some examples of applications involving clustering techniques are: image processing [8], analysis of gene expression data [39], studies of some diseases; e.g., cancer [26], topic detection and tracking [10, 32], intrusion detection [25], spatial data analysis [18,20,24], among others. Clustering is the process of grouping a collection of objects into a set of meaningful classes called *clusters*, so that objects belonging to the same cluster should be more similar than objects belonging to different clusters [21].

There are some applications, like medical diagnosis, categorization of digital libraries, topic detection and others, where some objects could belong to more than one cluster [3,5,33]. However, most clustering algorithms do not allow objects to belong to more than one cluster; i.e., they build disjoint clusters.

As an example where building non disjoint clusters is needed, consider an application for a health care facility where there are records describing the patient's symptoms. Since a patient could have more than one disease then, if clusters represent diseases, a patient could belong to more than one cluster. Similar examples can be found in tasks like: news stream analysis [33], text segmentation [1], among others.

*Corresponding author: Airel Pérez-Suárez, Computer Science Department, National Institute of Astrophysics, Optics and Electronics Luis Enrique Erro ♯1, Sta. María Tonantzintla, Puebla, CP: 72840, Mexico. E-mail: airel@inaoep.mx.

Clustering algorithms can be classified according to different criteria such as their capability to process changes that modify the dataset [15]. *Static algorithms* suppose that the entire dataset is available before clustering; therefore, when objects are added or deleted from the dataset, these algorithms must reprocess the whole dataset to build the set of clusters; i.e., they do not take advantage of the previous clusters. On the other hand, *incremental algorithms* are able to process new objects added to the datasets and consequently, they can update the set of clusters using the previous clusters. Finally, *dynamic algorithms*, in addition to incremental ones, are able to update the clustering when some objects are removed or modified (a modification can be viewed as a deletion followed by an addition and in this way, modifications will be viewed in this work). Most the clustering algorithms reported in the literature, are static. However, static clustering algorithms are useful only if it is granted that the dataset will not change anymore. In other cases, for environments like the World Wide Web, news streams and others, where the dataset changes frequently, static clustering algorithms become inefficient and dynamic algorithms are more suitable.

The main contribution of this paper is a new dynamic clustering algorithm, called DCS, which introduces a new graph-covering strategy that allows obtaining a set of clusters that could overlap. Additionally, a new strategy for efficiently updating the clustering after multiple object additions and/or deletions, is also introduced.

The experimental evaluation, conducted over different data collections, shows that our proposed algorithm is faster than the Star algorithm [4] (which is the unique algorithm that faces the problem of overlapping clustering in a dynamic context) for processing multiple additions and/or deletions, maintaining a comparable and even better clustering quality according to the Fmeasure [6] and Jaccard--index [23] evaluation measures. In addition, our proposed algorithm obtains fewer clusters, which is a desirable property in some real applications handling overlapping clustering such as information organization [3], filtering [5], web document clustering [19], among others.

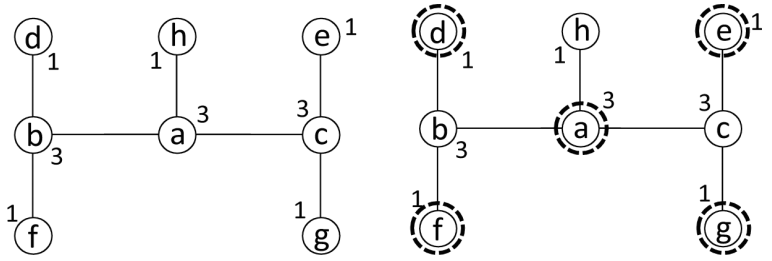
The remainder of this paper is organized as follows: in Section 2, the related work is outlined. In Section 3, we introduce the DCS algorithm. The experimental evaluation, showing the performance of the DCS algorithm on several document collections, is presented in Section 4. Finally, the conclusions and some ideas about future work are presented in Section 5.

2. Related work

There are different clustering algorithms, reported in the literature, which are able to work with datasets that could change over time [4,9,12,13,19,22,27,31,33,36,37,40]; however, most of these algorithms build disjoint clusters or they are incremental ; i.e., they only process additions.

From the set of algorithms that can deal either with additions or deletions [4,9,12,13], only the Star algorithm [4] faces the problem of overlapping clustering in a dynamic context, while Ant-Cluster [9], DB-Colc [12] and IncrementalDBSCAN [13] build disjoint clusters. Based on this fact, in this section we just describe in detail the Star algorithm, which will be used for comparing the behavior of the algorithm proposed in this work. The DHS algorithm reported in [15] was not included as related work because it faces the problem of hierarchical clustering in a dynamic context, which is out of the scope of this paper.

Although our work seems to be close to algorithms developed for clustering data streams [2,17,28], 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, like those introduced in [2,17,28], are just able to process additions. While, dynamic algorithms, like the one we introduce in this work, are able to process additions, deletions and



(a) A similarity graph G_β (b) Centers selected by the Star algorithm

Fig. 1. Illustrating one limitation of the Star algorithm.

modifications. Second, to the best of our knowledge, none of the algorithms developed for clustering data stream faces the problem of overlapping clustering, which is the problem addressed in this work. Based on the differences above explained, we decided do not include, as related work, algorithms facing the data stream clustering problem, like those proposed in [2,17,28].

The Star algorithm [4] is a *graph-based* clustering algorithm that has been used for filtering [5] and for information organization tasks [3]. Star considers the collection of objects represented by its associated *thresholded similarity graph* and, like many graph-based algorithms, it builds a clustering through a *cover* of this graph.

Given a collection of objects $O = \{o_1, o_2, \ldots, o_n\}$, a user-defined parameter β and a similarity function S such that $\forall o_i, o_j \in O, o_i \neq o_j$, $S(o_i, o_j) = S(o_j, o_i)$; a *thresholded similarity graph* is an undirected graph $G_\beta = \langle V, E_\beta \rangle$ where $V = O$ and $(o_i, o_j) \in E_\beta$ if and only if $S(o_i, o_j) \geqslant \beta$. A *star-shaped sub-graph* is a sub-graph of $m+1$ vertices, a special vertex c called *center* and m vertices called *satellites*, such that there is an edge between the center and each satellite. When a star-shaped sub-graph only contains the center then it is called *degenerated*. For the Star algorithm each star-shaped subgraph is interpreted as a cluster.

Star builds a cover of G_β using star-shaped sub-graphs. This cover is obtained through a *greedy* heuristic which selects in each iteration the most dense sub-graph; i.e., the sub-graph containing the highest number of satellites. When some objects are added or deleted from the collection, some star-shaped sub-graphs must be deleted, updated or created; therefore, the current clustering must be updated.

In order to update the clustering after a change, Star iteratively analyzes a list L initially containing all the satellites adjacent to the added, or removed, vertex. In each iteration, the vertex $v \in L$ having the highest degree is selected and if v has a degree greater than any of its adjacent centers then v is promoted to center and removed from L. After the promotion, each center c adjacent to v is removed from the list of centers and each vertex $u \in c.Adj$ is inserted into L. The aforementioned process finishes when L becomes empty.

The Star algorithm has two main limitations. First of all, it is not able to process either multiple additions or multiple deletions. It is important to clarify what we understand as *multiple* additions/deletions and why it is important to deal with them. Let's suppose that a set of objects O will be added to the dataset. The Star algorithm adds the objects of O to the dataset one by one, updating the clusters after each addition. Notice that, if more than one object is added to the same cluster then this strategy would consume a long time. A better choice would be to allow the algorithm to add all the objects to the dataset and after that to update the clusters that were affected by the additions. It is important to notice that the previous situation could also happen with deletions or with a combination of additions and deletions.

Another limitation of Star is that it tends to build a lot of clusters, each one with few objects. It is important to notice that, when a vertex v is selected as center, all its adjacent vertices are removed from

L. Let suppose that these removed vertices have a degree greater than any vertex remaining in L. Since the remaining vertices covers less vertices than the one covered by the removed vertices, it is possible that we need to select more centers than the ones we would need to select without removing the adjacent vertices of v. This situation is described through an example in Fig. 1; in this figure the vertices appear labeled with their degree. In Fig. 1, the vertices selected as center were highlighted using a dashed line.

As it can be seen in Fig. 1(b), when vertex a is selected as center, both vertices b and c are removed from the list of candidates. Therefore, Star will select vertices d, e, f and g as center for covering G_β. In this way, the number of clusters is greater than the one that would be obtained if vertices b and c were not removed from L and consequently, they were selected by the algorithm. As it will be showed later on in Subsection 3.2, our proposed algorithm overcomes this limitation.

The algorithm proposed in this work, named DCS, introduces a new graph-covering strategy that allows us to obtain a set of overlapping clusters, together with a strategy that allows DCS to update the clustering, by efficiently managing multiple additions and deletions.

3. Clustering based on *strength*

The presentation of the proposed dynamic clustering algorithm is divided into four subsections. First of all, in Subsection 3.1, we give some basic concepts needed for introducing our algorithm. Second, in Subsection 3.2 the new strategy proposed to cover G_β is introduced. Afterwards, in Subsection 3.3, we introduce the new strategy for updating a clustering when multiple objects are added and/or deleted, together with the pseudocode of the DCS algorithm. Finally, in Subsection 3.4 some characteristics of the DCS algorithm are discussed.

3.1. Basic concepts

Let $G_\beta = \langle V, E_\beta \rangle$ be a thresholded similarity graph and v a vertex of G_β; the *set of adjacent vertices* of v, denoted by *v.Adj*, is the set of vertices $u \in V$, such that there is an edge $(v, u) \in E_\beta$. The vertices having an empty set of adjacent vertices are known as *isolated*. Additionally, the cardinality of *v.Adj* is known as the *degree* of v.

Let $G_\beta = \langle V, E_\beta \rangle$ be a thresholded similarity graph; a *star-shaped sub-graph* (*s-graph*) in G_β is a sub-graph $G'_\beta = \langle V', E'_\beta \rangle$ such that $V' \subseteq V$, $E'_\beta \subseteq E_\beta$ and there is a vertex $c \in V'$ which meets $\forall v \in V', v \neq c$, there is an edge $(c, v) \in E'_\beta$. The vertex c is called the *center* of the s-graph and the remaining vertices are called *satellites*.

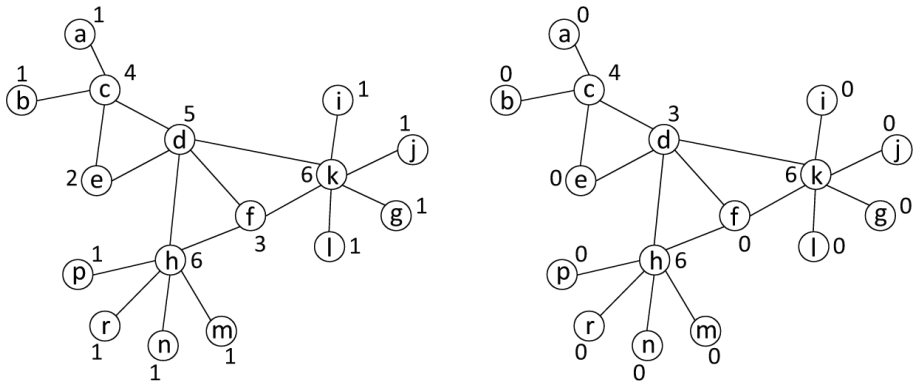
Let $W = \{G^1_\beta, G^2_\beta, \ldots, G^k_\beta\}$ be a set of s-graphs which were built from a thresholded similarity graph $G_\beta = \langle V, E_\beta \rangle$; the set W is a *cover* of G_β iff $\forall v \in V$, $\exists G^i_\beta = \langle V^i, E^i_\beta \rangle \in W$, such that $v \in V^i$.

Let $v, u \in V$ be two vertices of G_β and $G'_\beta = \langle V', E'_\beta \rangle$ the s-graph where v is the center; we say that v *covers* u iff $u \in V'$.

Let $G_\beta = \langle V, E_\beta \rangle$ be a thresholded similarity graph and $M = \{v_1, v_2, \ldots, v_k\}$ a set of vertices where $\forall i = 1..k, v_i \in V$; M is a *β-connected component* iff it meets the following conditions:

i) $\forall v_i, v_j \in M$, $v_i \neq v_j$, there are $v_{i_1}, v_{i_2}, \ldots, v_{i_q} \in M$, such that $\forall p = 1..q-1, (v_{i_p}, v_{i_{p+1}}) \in E_\beta$ and $v_{i_1} = v_i$ and $v_{i_q} = v_j$ or $v_{i_1} = v_j$ and $v_{i_q} = v_i$.
ii) There is not another set M', satisfying condition i), such that $M \subset M'$.

The set formed by a singleton *isolated* vertex of G_β is considered a *degenerated* β-connected component.



(a) Using the *degree* as label (b) Using the *strength* as label.

Fig. 2. Illustrating the use of the strength in the covering of a graph G_β.

3.2. Building overlapping clusters using strength

Finding the minimum vertex cover of a graph G_β is a NP complete problem [14]; however, we can build a cover of G_β, using s-graphs, that approximates the minimum cover. Since a s-graph is determined by its center, the problem of finding a set $W = \{G^1_\beta, G^2_\beta, \ldots, G^k_\beta\}$ of s-graphs, which covers G_β, can be seen as the problem of finding the set $C = \{c^1, c^2, \ldots, c^k\}$ where $\forall i = 1..k$, c^i is the center of the s-graph $G^i_\beta \in W$. Since each vertex in G_β forms a s-graph then, initially, all vertices are *candidates* to be included in C; therefore, it would be useful to define a criterion to reduce the number of candidates and also it is important to establish a selection order among the candidates. The proposed algorithm is based on a property of the vertices called *strength*.

The *strength* of a vertex $v \in V$, denoted by *v.strength*, is computed as follows:

$$v.strength = |\{w \in v.Adj \mid v.count \geqslant w.count\}|\,,$$

where *v.Adj* is the set of adjacent vertices of v and *v.count* is the number of vertices $z \in v.Adj$ which have a degree non greater than the degree of v; *w.count* is defined in the same way as *v.count*.

Ideally, iteratively selecting the vertex v with the highest degree (the one with the highest amount of adjacent vertices) will lead to cover G_β as fast as possible; however, this could not always so. If the vertex v, having the highest degree in the iteration i, is adjacent to d vertices which were selected in previous iterations then, the amount of vertices that could be included in the cover of G_β by vertex v will be $|v.Adj| - d$ instead of $|v.Adj|$ as it was supposed. Therefore, if there was a vertex u such that: (i) u is adjacent to q previously selected vertices and (ii) $|u.Adj| - q > |v.Adj| - d$ then, even when $|u.Adj| < |v.Adj|$, for covering G_β as fast as possible, selecting the vertex u in the iteration i would be a better choice than selecting v. By using the strength of the vertices instead of their degree in the covering process we can solve the above mentioned problem.

An example, illustrating the previous situation, is given in Fig.2. In this figure, we show a graph where the vertices appear labeled with their degree (see Fig. 2(a)) and a graph where the vertices appear labeled with their strength (see Fig. 2(b)). As it can be seen from Fig. 2(a), after the selection of vertices h and k (the vertices having the greatest degree) the vertex d is the remaining vertex having the highest degree; however, if we select vertex d we still need to select vertex c to entirely cover the graph. On the contrary, as it can be seen from Fig. 2(b), this problem does not appear when the strength is used, since vertex c has a strength greater than vertex d.

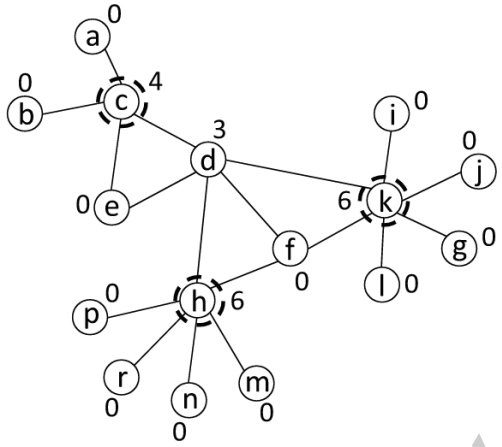


Fig. 3. Centers selected for covering the graph of Fig. 2(b).

Intuitively, *v.strength* measures the amount of vertices that vertex v could include in the cover of G_β if all the vertices, which could include more vertices than v, were selected before v.

From the previous analysis, we can conclude that the only vertices that need to be verified are those having a strength greater than zero, besides the vertices should be selected in decreasing order according to their strength value. All vertices having a strength greater than zero are added to a *list of candidates* L. The isolated vertices of G_β are included directly in C.

The candidates in the list L are analyzed in descending order according to the strength of the vertices; in this way, the number of s-graphs needed to cover G_β could be reduced. Each vertex $v \in L$ is selected as a center and added to C if it satisfies one of the following conditions:

1) v has not been covered yet; i.e., $v \notin C$ and $\nexists\, c \in C$, such that $(c, v) \in E_\beta$.
2) v is already covered by other vertices in L but it has at least one adjacent vertex which is not covered yet. This condition avoids selecting centers that have all their satellites covered by previous selected centers; i.e., centers that do not help to cover G_β.

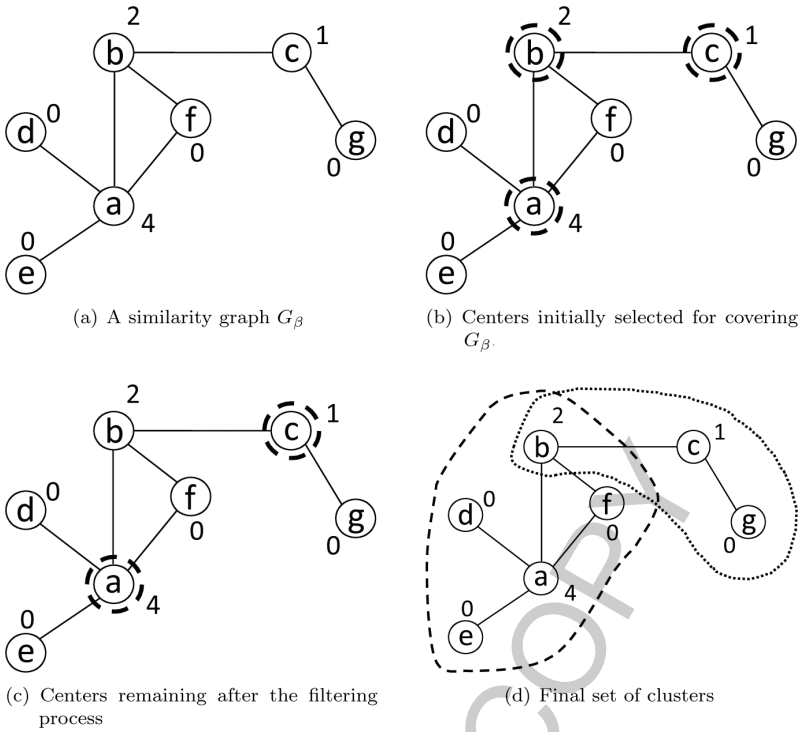
In Fig. 3, we show how the graph of Fig. 2(b) is covered using the above explained strategy. In this figure, the vertices selected as center are highlighted using a dashed line.

Once the set C is built, there could be some vertices belonging to C which are not useful to cover G_β; i.e., they could be removed from C and G_β would still be covered. In order to remove from C all the centers that are no longer useful to cover G_β, some additional steps must be done. First of all, the set C is sorted in ascending order according to the degree of the centers; after that, all *redundant* centers are removed from C. A center $c \in C$ is redundant if it satisfies the following conditions:

a) c has an adjacent center whose degree is greater than the degree of c.
b) Each vertex $u \in c.Adj$ is a center or it has, at least, another adjacent center different from c.

Finally, after removing all redundant centers, the s-graphs formed by the centers in C are the final clusters.

In Fig. 4, we illustrate all the above explained ideas using an example. Fig. 4(a) shows a similarity graph where all vertices are labeled with their strength. Figure 4(b) shows the centers that are selected by our algorithm; the vertices selected as center are highlighted using a dashed line. In Figure 4(c) we show the remaining centers after removing redundant centers. Finally, in Fig. 4(d) we showed the final clusters obtained by the above process.



(a) A similarity graph G_β

(b) Centers initially selected for covering G_β.

(c) Centers remaining after the filtering process

(d) Final set of clusters

Fig. 4. Illustrating the step for covering a graph G_β.

The above described strategy constitutes a static algorithm for clustering data. The pseudocode of this algorithm is showed in Algorithm 1.

Algorithm 1: Covering G_β using *strength*

Input: $O = \{o_1, o_2, \ldots, o_n\}$ - a collection of objects, β - a similarity threshold

Output: SC - a set of clusters

```
1  "Build G_β from collection O using the similarity threshold β";
2  "Calculate the strength of each vertex in G_β";
3  C := {v ∈ V | v.Adj = ∅};
4  L := {v ∈ V | v.strength > 0};
5  "Sort L in descending order by strength";
6  foreach vertex v ∈ L do
7  |  if v satisfies condition 1) or 2) then C := C ∪ {v};
8  end
9  "Sort C in ascending order by degree";
10 "Remove from C all redundant centers";
11 SC = ∅;
12 foreach center c ∈ C do
13 |  SC := SC ∪ {{c} ∪ c.Adj};
14 end
```

It is important to notice that the strategy proposed in this subsection for covering G_β allows centers to be adjacent each other. Besides, since we selected the vertices in descending order according to

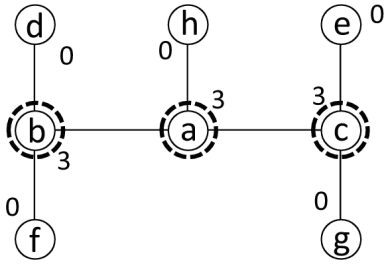


Fig. 5. Centers selected for the graph of Fig. 1(a) following our proposed strategy.

their strength and we remove any unnecessary cluster, we obtain fewer clusters than the Star algorithm. Following the ideas presented in this subsection, the graph of Fig. 1(a) can be covered using fewer centers, as it is showed in Fig. 5. In this figure, the vertices selected as centers were highlighted using a dashed line.

For determining the computational complexity of Algorithm 1, we will analyze each one of its steps. Let n be the size of the object collection $O = \{o_1, o_2, \ldots, o_n\}$; i.e., $|O| = n$. In Step 1, the similarity between all pairs of objects is calculated. Therefore, the time spent in this step is $T_1 = n^2$, thus T_1 is $O(n^2)$.

In Step 2, in order to compute the strength of each vertex, *v.count* must be calculated and then, using it, *v.strength* is computed. Based on the definition of strength the time spent by Step 2 is $T_2 = 2 \cdot n^2$; i.e., the computational complexity of Step 2 is $O(n^2)$.

Steps 3 and 4 can be executed by checking the degree and the strength of all vertices in V. Thereby, the time spent by these steps is $T_3 = n$; thus, T_3 is $O(n)$. Since the strength of a vertex v is an integer in [0,$n-1$], following the *pigeonhole principle*, the sorting process of Step 5 can be done in a time $T_4 = n$; hence, T_4 is $O(n)$.

In order to verify if a candidate $v \in L$ satisfies condition 1 or condition 2 (see Steps 6–8) it is necessary to visit all vertices in *v.Adj*; thus, the time of this process is $T_5 = n^2$. Given that T_5 is $O(n^2)$, the computational complexity of Steps 6–8 is $O(n^2)$.

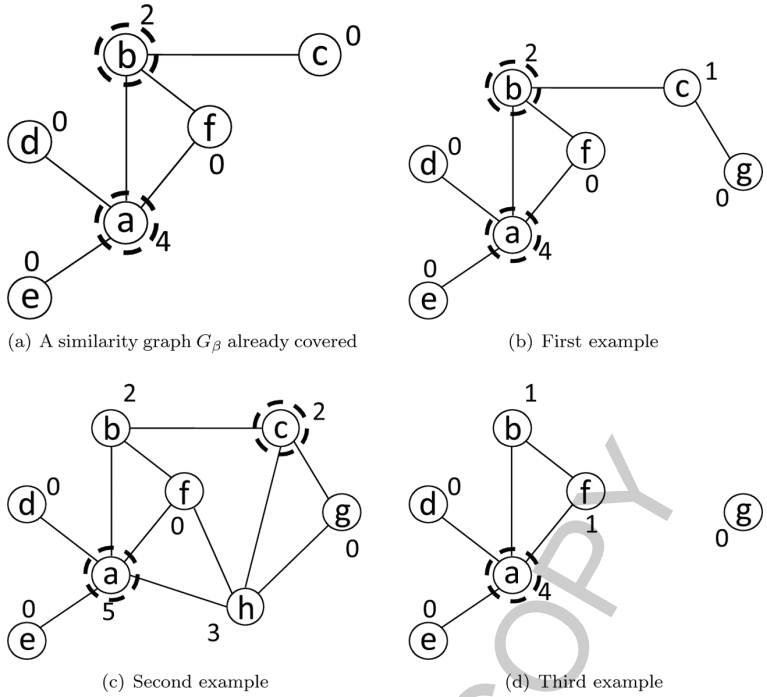
Step 9, following the same explanation of Step 5, spends a time $T_6 = n$; therefore, T_6 is $O(n)$. For removing the redundant centers it is necessary to verify, for each center $c \in C$, the conditions a and b; therefore, the time spent in this process is $T_7 = |C| \cdot n$. Since, in the worst case, $|C| = n$, then $T_7 = n^2$, thus T_7 is $O(n^2)$. As the final clusters are the s-graphs formed by each center $c \in C$, the set SC can be built in a time $T_8 = n^2$; thus T_8 is $O(n^2)$.

Finally, the time of the entire algorithm is $T_t = \sum_{i=1}^{8} T_i$, where $T_i, i = 1..8$, are the times of each step of the algorithm. By the rule of the sum, T_t is $O(\max(T_i \mid i = 1..8))$. From the the previous analysis we can conclude that the complexity of Algorithm 1 is $O(n^2)$.

3.3. Updating the current clustering

Let's suppose that we have a similarity graph $G_\beta = \langle V, E_\beta \rangle$ covered using Algorithm 1 and that C is the list of centers covering G_β. In order to update the current clustering when there are changes in the collection, it is important to know, first of all, how these changes could affect the current cover of G_β. In the analysis presented in this subsection, it will be assumed that: (i) more than one object could be added to/deleted from G_β at the same time, and (ii) it could be a combination of both operations.

It is important to mention that each addition or deletion impacts the topology of G_β. The addition of an object to the collection implies the addition of a new vertex in G_β, and consequently to compute its



(a) A similarity graph G_β already covered (b) First example

(c) Second example (d) Third example

Fig. 6. Illustrating the situations that could happen when vertices are added to/removed from the graph.

similarity to all the other vertices in G_β. On the other hand, when an object is deleted from the collection, the vertex that represents this object together with its associated edges are removed from G_β. In both cases, the strength of some vertices in G_β must be recomputed.

After some vertices are added to/removed from G_β there are two situations that could happen:

a) Some vertices become uncovered. This situation happens when at least one added vertex does not have any center in its set of adjacent vertices, or when all the centers covering a specific vertex are deleted.
b) The strength of some vertices changes and, as a consequence of that, there is at least one vertex v which has a strength greater than: (i) at least one of the centers in *v.Adj* or (ii) at least one center covering vertices in *v.Adj*.

In Fig. 6, we showed an example where situations a) and b) happen. In Fig. 6(a), we showed an already covered graph; in this graph, the selected centers are highlighted using dashed lines. In Fig. 6(b), we showed the graph resulting from adding the vertex g to the graph of Fig. 6(a). As it can be seen from Fig. 6(b), the vertex g is uncovered and therefore the vertex c should be selected as a center for updating the covering. After updating the covering of the graph in Fig. 6(b) if we add the vertex h then, we obtain the graph of Fig. 6(c). Although all vertices of Fig. 6(c) are covered, the vertex h has higher strength than vertex c therefore, vertex h should be selected as a center instead of vertex c. On the other hand, after updating the covering of the graph in Fig. 6(b), if we remove the vertex c then, we obtain the graph of Fig. 6(d) . As it can be seen from this graph, the vertex g became uncovered.

In order to update the clustering when situation a) happens some new centers must be selected and included in C; therefore, for each uncovered vertex v one vertex $u \in$ *v.Adj* must be selected as a center.

Situation b), on the other hand, motivates a deeper analysis. A vertex v could change its strength if at least one of the following conditions happens:

i) The value of *v.count* changes.
ii) There is at least one vertex $u \in$ *v.Adj* such that the value of *u.count* changes.

Following a similar reasoning, a vertex v could change its *v.count* only if at least one of the following conditions happens:

i) One or more vertices are added to or removed from *v.Adj*.
ii) There is at least one vertex $u \in$ *v.Adj* such that one or more vertices are added to or removed from *u.Adj*.

From the previous analysis we can conclude that the vertices that could change their strength belong to the β-connected components that contain:

a) The vertices which were added to G_β.
b) The vertices which were adjacent to the vertices removed from G_β.

Therefore, we can update a clustering or covering of G_β by updating the covering of each β-connected component containing vertices that change their strength. Let $G' = \langle V', E' \rangle$ be one of those β-connected components and $C' \subseteq V'$ the list of centers covering G'. For updating the covering of G', first of all, the strength of each vertex in V' must be recomputed. Afterwards, the list L' of vertices $v \in V'$ that are candidates to be promoted to centers must be computed.

Let $V'_s \subseteq V'$ be the set of non centers vertices having a strength greater than zero; In order to compute the list L', the vertices $v \in V'_s$ and the vertices $c \in C'$ must be processed.

Each vertex $v \in V'_s$ fulfilling at least one of the following conditions will be considered as candidate, and consequently it will be included in L':

i) v is uncovered.
ii) v has at least one adjacent vertex which is not covered.
iii) v has at least one adjacent vertex u such that the center w, adjacent to u and having the greatest strength among all the adjacent centers of u, meets the condition that *v.strength* $>$ *w.strength*. In addition, all vertices like u are marked as *activated*; the vertices marked as *activated* are used during the analysis of the list C'.

In the analysis of the list C', the adjacent vertices of each center $c \in C'$ are visited. The vertices $v \in$ *c.Adj*, such that $v \notin C'$ and *v.strength* $>$ *c.strength*, are included in L'; all centers c having at least one adjacent vertex v, such that $v \notin C'$ and *v.strength* $>$ *c.strength*, are marked as *weak*. Once all the adjacent vertices of c have been visited, if c is marked as *weak* or it has at least one satellite marked as *activated* then, c is removed from C' since it could be replaced by other vertices having a greater strength. Finally, if *c.strength* > 0 then c is considered as a candidate and it is included in L'.

Once the candidate list L' is built, the cover of G' is updated using the same cover strategy used in the Steps 5–10 of Algorithm 1. Afterwards, each vertex in the β-connected component is marked as "processed" in order to guarantee that a vertex will not be processed more than once.

The above described strategy constitutes the DCS algorithm. The pseudocode of DCS is showed in Algorithm 2.

Notice that, unlike previous algorithms, the DCS algorithm processes first all the changes in G_β and after that it updates the current cover; thus, if there are two or more changes, additions or deletions that were done to G_β, affecting the same clusters, then these clusters will be updated just once instead of processing each change one by one as Star does. This last characteristic of DCS allows the algorithm to save time being able to efficiently manage multiple additions and/or deletions of objects.

Algorithm 2: DCS algorithm

Input: $G_\beta = \langle V, E_\beta \rangle$ - a thresholded similarity graph, β - a similarity threshold, R^+ - set of added objects, R^- - set of objects to be removed

Output: $G_\beta = \langle V, E_\beta \rangle$ - the updated thresholded similarity graph, SC - a set of clusters

```
1  M := ∅;
2  foreach vertex u ∈ R⁻ do
3  |  M := M ∪ (u.Adj \ R⁻);
4  |  "Remove u from G_β";
5  end
6  foreach vertex v ∈ R⁺ do
7  |  M := M ∪ {v};
8  |  "Add v to G_β";
9  end
10 foreach vertex v ∈ M do
11 |  if v is not-processed then
12 |  |  "Build the β-connected component G' = ⟨V', E'⟩ of v";
13 |  |  if G' is an isolated vertex then "Mark v as center";
14 |  |  else
15 |  |  |  "Update strength for vertices in G'";
16 |  |  |  "Build V'_s, C' and then the candidate list L'";
17 |  |  |  "Sort L' in descending order by strength";
18 |  |  |  foreach vertex v ∈ L' do
19 |  |  |  |  if v satisfies condition 1) or 2) then C' := C' ∪ {v};
20 |  |  |  end
21 |  |  |  "Sort C' in ascending order by degree";
22 |  |  |  "Remove from C' all redundant centers";
23 |  |  end
24 |  |  "Mark vertices in G' as processed";
25 |  end
26 end
27 "Mark vertices in G_β as not-processed";
28 SC = ∅;
29 foreach vertex v ∈ V marked as center do
30 |  SC := SC ∪ {{v} ∪ v.Adj};
```


For determining the computational complexity of the DCS Algorithm, we will analyze each one of its steps. Let n be the number of vertices of G_β in any step of the algorithm; i.e., $n = |V|$.

Steps 2–9 update G_β by processing first the set R^- and then the set R^+; this order in the processing of the added and/or removed objects avoids unnecessary calculation of the similarity between objets in R^+ and objets in R^-. Steps 2–5 spend a time $T_1 = \frac{n \cdot (n-1)}{2}$; thus, T_1 is $O(n^2)$. Steps 6–9 spend a time $T_2 = n^2$, then T_2 is $O(n^2)$.

The time spent by Steps 10–26 depends on the time spent by Steps 12–24. The construction of $G' = \langle V', E' \rangle$ in Step 12 can be done in a time $T_3 = n_i^2$, where $n_i = |V'|$. The time spent by Steps 13–23 depends, in the worst case, on the time spent by Steps 15–22. Step 15, as explained for Algorithm 1, spends a time $T_4 = 2 \cdot n_i^2$. The construction of L' in Step 16 can be done in $T_5 = n_i^2$. The time of Steps 17–22, as explained for Algorithm 1, is $T_6 = 2 \cdot n_i^2 + 2 \cdot n_i$ and the time of Step 24 is $T_7 = n_i$.

Based on the previous analysis the time of Steps 11–25 is $T_{11-25} = 6 \cdot n_i^2 + 3 \cdot n_i$. Let $M = \{v_1, v_2, \ldots, v_k\}$ be the set of vertices for which Steps 10–26 are executed. If the size of the $\beta-$connected

component $G' = \langle V', E' \rangle$ generated by each vertex v_i is n_i then the time spent in Steps 10–26 is:

$$T_{10-26} = \sum_{i=1}^{k} (6 \cdot n_i^2 + 3 \cdot n_i) = \sum_{i=1}^{k} 6 \cdot n_i^2 + \sum_{i=1}^{k} 3 \cdot n_i = 6 \cdot \sum_{i=1}^{k} n_i^2 + 3 \cdot \sum_{i=1}^{k} n_i \quad (1)$$

Given that $\sum_{i=1}^{k} n_i = n$, where $n = |V|$, then substituting in Eq. (1):

$$T_{10-26} = 6 \cdot \sum_{i=1}^{k} n_i^2 + 3 \cdot \sum_{i=1}^{k} n_i = 6 \cdot \sum_{i=1}^{k} n_i^2 + 3 \cdot n$$

$$T_{10-26} \leqslant 6 \cdot (\sum_{i=1}^{k} n_i)^2 + 3 \cdot n \leqslant 6 \cdot n^2 + 3 \cdot n$$

then, the computational complexity of Steps 10–26 in the worst case is $O(n^2)$. The time spent by Step 27 is $T_8 = n$ and the time spent by Steps 28–30 is, in the worst case, $T_9 = n^2$; therefore, T_8 is $O(n)$ and T_9 is $O(n^2)$.

Finally, based on the aforementioned analysis the total time spent by the DCS algorithm is $T_t = T_1 + T_2 + T_{10-26} + T_8 + T_9$. By the rule of the sum T_t is $O(\max(T_1, T_2, T_{10-26}, T_8, T_9))$; therefore, T_t is $O(n^2)$.

3.4. Final considerations about the DCS algorithm

Our proposed algorithm, unlike other dynamic algorithms, is able to process multiple additions and/or deletions. That is, when a set of additions and/or deletions must be processed at a given time, instead of updating the clustering after each change, our algorithm process all changes and after that, the clustering is updated. In this way, when some addition and/or deletion operations affect the same clusters, DCS saves processing time because it update those clusters just once rather than several times.

Finally, as it can be noticed from Algorithm 2, the DCS algorithm supposes that there exists a graph G_β, representing the current collection, which were previously covered. However, if there is no previous collection and it is the first time that the collection will be clustered then, the graph G_β needed as input parameter is an empty graph; in this way, DCS can process a collection without the existence of a previous clustering.

4. Experimental results

In order to show the performance of the proposed algorithm, some experiments were done over several overlapping collections. The experiments were divided into three types: those for comparing the algorithms according to the quality of the clustering, those for evaluating the time spent by each algorithm for processing of multiple additions and/or deletions, and finally, those for comparing the algorithms according to the number of clusters.

In all these experiments we contrast our results against those obtained by the Star algorithm [4]. We used Star in our experiments because it is the unique clustering algorithm facing the problem of overlapping clustering in a dynamic context. Both the Star and DCS algorithms were implemented in C++. All the experiments presented in this subsection 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.

Table 1
Characteristics of the document collections

Collection	#Documents	#Classes	#Terms
AFP	695	25	11785
Reu-Te	3587	100	15113
Reu-Tr	7780	115	21901
Reuter	11367	120	27083
TDT	16007	193	68019
TDT-1	8602	176	51764
TDT-2	7404	178	44610
TDT-3	10258	174	53706
TDT-4	10074	172	53036
TDT-5	11328	182	55923

4.1. Description of the collections

Since we are facing the problem of overlapping clustering, we decided to evaluate the algorithms in the task of document clustering, where it is common for a document to belong to more than one topic.

The document collections used in the experiments were built from three benchmark text collections: AFP, Reuters-21578 and TDT2. The AFP benchmark was downloaded from http://trec.nist.gov and it contains news published by the AFP agency in 1994 and used in the TREC-5 conference. Reuters--21578 was downloaded from http://kdd.ics.uci.edu and it contains news published by Reuters during 1987. TDT2 was downloaded from http://www.nist.gov/speech/tests/tdt.html ant it contains news stories collected from different sources from January 1998 to June 1998. AFP contains news in Spanish while the other two benchmarks contain news in English.

Ten document collections were built from the aforementioned benchmarks: (1) AFP was built from the AFP benchmark using all its news, (2) Reu-Te was built from Reuters benchmark using the news tagged as "Test" that have been associated with at least one topic, (3) Reu-Tr was built from Reuters benchmark using the news tagged as "Train" that have been associated with at least one topic, (3) Reuters, is the union of Reu-Te and Reu-Tr, (4) TDT was built from TDT2 benchmark using the news that have been associated with at least one topic and (5) five sub-collections of TDT called as TDT-1, TDT-2, TDT-3, TDT-4 y TDT-5. In order to build these five sub-collections of TDT, the news contained in the TDT collection were randomly arranged into 5 folds. Afterwards, each sub-collection was built by selecting randomly three from the five folds.

The characteristics of the ten document collections are shown in Table 1. In Table 1, the column labeled as "Classes" corresponds to the number of topics or classes which were manually identified by experts for each document collection. The ground truth of each collection is distributed together with the collection.

In our experiments, documents were represented using the Vector Space Model (VSM) [35]. The index terms of the documents represent the lemmas of the words occurring at least once in the whole collection; these lemmas were extracted from the documents using the 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 normalized by the logarithm [16]. The cosine measure was used to calculate the similarity between two documents [7].

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

4.2. Evaluation measures

There are several measures proposed to evaluate the quality of the clusters obtained by a clustering algorithm [29]. Most of these measures were developed to evaluate disjoint clusters; i.e., they are not useful to evaluate clustering algorithms in scenarios where objects can belong to more than one cluster. In our experiments, we use the Fmeasure [6] and Jaccard-index [23] which are external measures used to evaluate overlapping clusterings [15,30].

We do not use internal measures like the Dunn index [11] and the silhouette width [34], because this type of evaluation measures has not been defined for evaluating overlapping clusters, which is the problem addressed in our paper. Moreover, a deep study is required in order to extend this type of measures for evaluating overlapping clusters.

Both, Fmeasure and Jaccard-index, are external measures which evaluate quality based on how much the set of clusters obtained by the algorithm resembles the set of classes manually labeled by experts; the higher the value of each measure the better the clustering. These two measures evaluate the quality of a clustering using statistics over pairs of objects.

Let $G = \{g_1, g_2, \ldots, g_k\}$ be the clusters obtained by a clustering algorithm and $C = \{c_1, c_2, \ldots, c_k\}$ be the set of classes in which, according to the judgment of the experts, the collection should be clustered.

Let n_{11} be the number of pairs of objects belonging to the same cluster and class, n_{10} be the number of pairs belonging to the same cluster but different class and n_{01} be the number of pairs belonging to the same class but different cluster.

Given a set of clusters like G and a set of classes like C, the expressions that define the Jaccard-index (J) and Fmeasure (F) are:

$$J(G,C) = \frac{n_{11}}{n_{10} + n_{11} + n_{01}} \tag{2}$$

$$F(G,C) = 2 \cdot \frac{prec(G,C) \cdot rec(G,C)}{prec(G,C) + rec(G,C)} \tag{3}$$

where:

$$prec(G,C) = \frac{n_{11}}{n_{11} + n_{10}} \text{ and } rec(G,C) = \frac{n_{11}}{n_{11} + n_{01}}$$

Both measures take values in [0,1] and, as it was mentioned before, the closer to 1 the value of each measure is, the better the clustering is. Besides, Fmeasure and Jaccard-index measures take into account the *Homogeneity* and *Completeness* of a clustering.

Homogeneity states that the clusters should not mix objects from different classes. Notice from Eq. (2) that building a cluster of objects belonging to different classes increases the value of n_{10} and then the value of $J(G,C)$ will decrease; therefore, Jaccard-index takes into account the Homogeneity of clusters.

In order to show that Fmeasure takes into account the homogeneity of clusters, $prec(G,C)$ and $rec(G,C)$ are substituted in Eq. (3):

$$F(G,C) = 2 \cdot \frac{\frac{n_{11}}{n_{11}+n_{10}} \cdot \frac{n_{11}}{n_{11}+n_{01}}}{\frac{n_{11}}{n_{11}+n_{10}} + \frac{n_{11}}{n_{11}+n_{01}}} = 2 \cdot \frac{\frac{n_{11}^2}{(n_{11}+n_{10})(n_{11}+n_{01})}}{\frac{n_{11}\cdot(n_{11}+n_{01})+n_{11}\cdot(n_{11}+n_{10})}{(n_{11}+n_{10})(n_{11}+n_{01})}}$$

$$F(G,C) = \frac{2 \cdot n_{11}^2}{n_{11} \cdot (n_{11} + n_{01}) + n_{11} \cdot (n_{11} + n_{10})} = \frac{2 \cdot n_{11}^2}{n_{11} \cdot (2 \cdot n_{11} + n_{10} + n_{01})}$$

thus:

$$F(G,C) = \frac{2 \cdot n_{11}}{2 \cdot n_{11} + n_{10} + n_{01}} \tag{4}$$

It is easy to see from Eq. (4) that, if the value of n_{10} increases then the value of F(G,C) will decrease; therefore, Fmeasure takes into account the homogeneity in a clustering.

Completeness, on the other hand, states that objects belonging to the same class should be grouped together in the same cluster. If in a clustering, the objects belonging to the same class are clustered in two or more clusters, then the value of n_{11} decreases and the value of n_{01} increases; hence, the value of $J(G,C)$ will decrease, showing in this way that Jaccard-index takes into account the completeness of a clustering.

Working on Eq. (4) we have:

$$F(G,C) = \frac{2 \cdot n_{11}}{2 \cdot n_{11} + n_{10} + n_{01}} = \frac{1}{\frac{2 \cdot n_{11} + n_{10} + n_{01}}{2 \cdot n_{11}}}$$

and then:

$$F(G,C) = \frac{1}{1 + \frac{n_{10}}{2 \cdot n_{11}} + \frac{n_{01}}{2 \cdot n_{11}}} \tag{5}$$

As it can be noticed from Eq. (5), if the value of n_{11} decreases and the value of n_{01} increases then both $\frac{n_{10}}{2 \cdot n_{11}}$ and $\frac{n_{01}}{2 \cdot n_{11}}$ will increase and therefore, the value of $F(G,C)$ will decrease, showing in this way that Fmeasure takes into account the completeness of a clustering.

Notice that both properties, homogeneity and completeness, are basic goals that a clustering algorithm should accomplish: a clustering algorithm should keep objects from the same class together and objects from different classes apart; thus, by using Fmeasure and Jaccard-index we will be able to evaluate the quality of the resulting clusters.

4.3. Quality of the resulting clusters

In this experiment we compare the algorithms according to the quality, considering Jaccard-index and Fmeasure, of the clustering they build. This experiment was conducted as follows.

First of all, since both algorithms depend on the data order, we built for each document collection C, twenty collections $C_1, C_2, \ldots, C_{20}$ in such a way that all these collections are different wrt. the order of the documents. After that, we executed each algorithm over the twenty document collections of each collection C using values of β in [0.15,0.40] with an increment of 0.01; that is, we use $\beta = 0.15$, 0.16, 0.17 an so on.

After that, we computed the values of Jaccard-index and Fmeasure obtained by each algorithm on each execution. In these experiments, we realized that for values of β, greater than 0.40 and smaller than 0.15, the quality of the clustering decreases. For this reason, we do not use values of β out of the above mentioned interval.

Then, we computed the average value of Fmeasure and Jaccard-index obtained by each algorithm, over the twenty document collections of each collection C, for each value of β used in the above mentioned executions. It is important to mention that, even when both algorithms depend on the data order, the standard deviation of the values of Fmeasure and Jaccard-index obtained by each algorithm, over the

Table 2
Best average quality values obtained by each algorithm over each document collection

	AFP		Reu-Te		Reu-Tr		Reuter		TDT	
Alg.	F	J	F	J	F	J	F	J	F	J
Star	0.73	0.57	0.59	0.42	0.56	0.39	0.57	0.40	0.40	0.25
DCS	0.76	0.63	0.64	0.47	0.58	0.41	0.58	0.41	0.48	0.32
	TDT-1		TDT-2		TDT-3		TDT-4		TDT-5	
Alg.	F	J	F	J	F	J	F	J	F	J
Star	0.39	0.24	0.45	0.29	0.46	0.30	0.47	0.31	0.44	0.28
DCS	0.47	0.30	0.53	0.36	0.53	0.36	0.55	0.38	0.52	0.35

Table 3
Values of β where each algorithm obtains its best average performance considering the Jaccard-index and the Fmeasure

	AFP	Reu-Te	Reu-Tr	Reuter	TDT
Star	0.25	0.24	0.24	0.21	0.29
DCS	0.25	0.25	0.24	0.25	0.32
	TDT-1	TDT-2	TDT-3	TDT-4	TDT-5
Star	0.30	0.29	0.29	0.29	0.30
DCS	0.32	0.32	0.32	0.30	0.31

Table 4
Percent improvements obtained by the DCS algorithm over the Star algorithm

	Collections				
Measure	AFP	Reu-Te	Reu-Tr	Reuter	TDT
F	4.11	10.34	3.57	1.75	20.0
J	7.02	11.90	5.13	2.50	28.0
	Collections				
Measure	TDT-1	TDT-2	TDT-3	TDT-4	TDT-5
F	20.51	17.78	15.22	17.02	18.18
J	25.0	24.14	20.0	22.58	25.0

twenty document collections of each collection C, was smaller than 0.01 for each value of β used. This means that the Fmeasure and the Jaccard-index vary only a little for different orders.

Table 2 shows the best average values of Fmeasure (F) and Jaccard-index (J) obtained by each algorithm over each collection C. Table 3 shows the values of β where each algorithm obtained, for each document collection, the best average values of Jaccard-index and Fmeasure.

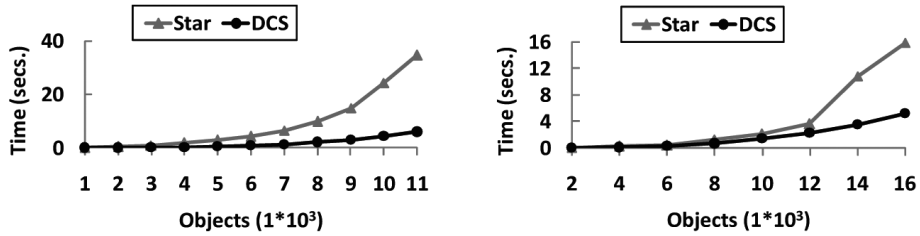
As it can be noticed from Table 2, DCS outperforms the Star algorithm, considering both evaluations measures, in all the collections used in this experiment.

In addition, we show in Table 4 the improvement in percentage of the values of Fmeasure (F) and Jaccard-index (J) obtained by our proposed algorithm wrt. the values obtained by Star considering the same evaluation measures.

As it can be seen from Table 4, DCS gets improvements up to the 20.51% considering the Fmeasure and the 28.0% considering the Jaccard-index.

4.4. Time spent for processing multiple operations

In these experiments we measure the time spent by each algorithm when multiple additions/deletions are done over the two largest collections; i.e., Reuters and TDT.



(a) Reuters, N_a=1000 (b) TDT, N_a=2000

Fig. 7. Behavior of the Star and DCS algorithms over Reuters and TDT for processing N_a multiple additions.

The experimental results presented in this subsection are divided into three parts. In the first part, the performance of each algorithm considering just multiple additions is presented. The second part is focused on measuring the time spent by each algorithm when multiple deletions are done over the collection and finally, the third part shows the behavior of each algorithm for multiple deletions followed by multiple additions.

4.4.1. Behavior for multiple additions

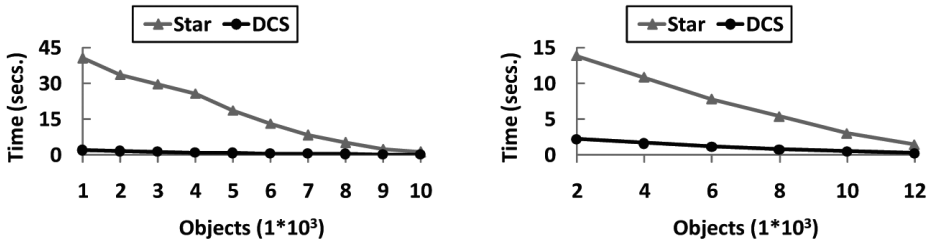
In this experiment, we measure the time that Star and DCS spent for updating the clustering every time a number N_a of documents is added to each collection. When the first N_a documents are added to the collection there is no previous clustering; therefore, these first N_a of added documents are clustered by both algorithms from the beginning. From this point, every time N_a documents are added to the collection, both methods update the previous clustering.

In the experiment with the Reuters collection we used for N_a the value of 1000 (see Fig. 7(a)). In the experiment with the TDT collection we used for N_a the value of 2000 (see Fig. 7(b)). The values of N_a were chosen taking into account the size of each collection. It is important to notice, that a fair comparison is only possible if both algorithms use the same value of β when they are processing the same collection. The use of different values of β would produce different thresholded similarity graphs. Therefore, we will not know if an algorithm has the best behavior because it has the best strategy or because it processes a graph different from the one processed by the other algorithm. Therefore, in this experiment we will use $\beta = 0.25$ and $\beta = 0.30$ for Reuters and TDT respectively.

In order to do a fair comparison, we built from each document collection ten different collections such that they are different wrt. the order of the documents they contain. After that, we executed both algorithms over all the collections created from Reuters and also over all the collections created from TDT. In Fig. 7, the average performance of the Star and DCS algorithms over Reuters and TDT, for the selected values of N_a, is shown.

As it can be seen from Fig. 7, our proposed algorithm has better average performance than Star when multiple additions are processed over the Reuters and TDT collections. We conducted other experiments using $N_a = 1500$ and $N_a = 2000$ for Reuters; and $N_a = 3000$ and $N_a = 4000$ for TDT. Using these values for N_a, we observed a behavior similar to that observed in the experiment above described.

From these experiments we realized that the order in which the documents are clustered and the size of the increment both have influence in the time spent by each algorithm for clustering the entire collection. That is, the time spent by the algorithm for processing different orders and different increments could be very different, depending on how difficult was to update the clustering after the changes. For example, we noticed that the time spent by the Star algorithm for clustering the TDT collection varies too much from one order to another when we use $N_a = 2000$ and $N_a = 3000$. However, the time spent by our



(a) Reuters, N_d=1000 (b) TDT, N_d=2000

Fig. 8. Behavior of the Star and DCS algorithms over Reuters and TDT for processing N_d multiple deletions.

algorithm for clustering TDT using $N_a = 2000$ and $N_a = 3000$, does not vary to much from one order to another.

4.4.2. Behavior for multiple deletions

In these experiments, we measured the time spent by the Star and DCS algorithms for updating the clustering built for Reuters and TDT, every time a number N_d of documents is randomly removed. We used for N_d the same values used for N_a in the experiment of Subsection 4.4.1; that is, $N_d = 1000$ for Reuters and $N_d = 2000$ for TDT. For building the previous clustering for the Reuters and TDT collections we used the same values of β chosen in Subsection 4.4.1; i.e., $\beta = 0.23$ for Reuters and $\beta = 0.30$ for TDT.

In order to do a fair comparison, in the same way as for multiple additions, we repeated this experiment ten times for each value of N_d, for both collections. In Fig. 8, the average performances of the Star and DCS algorithms, over Reuters and TDT, for the selected values of N_d, are shown.

As it can be seen from Fig. 8, our proposed algorithm clearly overcomes Star in the processing of multiple deletions over the Reuters and TDT collections. We conducted other experiments using $N_d = 1500$ and $N_d = 2000$ for Reuters; and $N_d = 3000$ and $N_d = 4000$ for TDT. Using these values for N_d, we observed a behavior similar to that observed in the experiment above described.

4.4.3. Behavior for multiple modifications

In these experiments, we measured the time each algorithm spent for multiple modifications. It is important to remember that both the Star and the DCS algorithms process a modification by processing a deletion followed by an addition.

First of all, we measured the time Star and DCS spent, for updating the clustering built for the Reuters and TDT , when a number N_m of documents is randomly deleted and added again but increasing or decreasing the weight of some terms belonging to the documents (i.e., modified).

We used for N_m the same values used in the previous experiments for N_a and N_d; that is, $N_m = 1000$ for Reuters and $N_m = 2000$ for TDT. For building the previous clustering for Reuters and TDT we used the same values of β chosen in Subsection 4.4.1; i.e., $\beta = 0.23$ for Reuters and $\beta = 0.30$ for TDT.

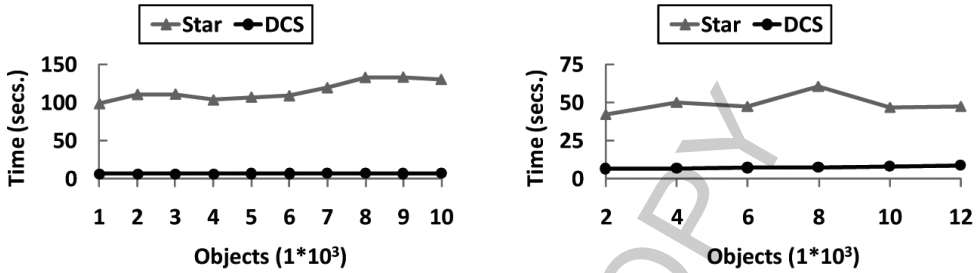
In order to do a fair comparison, in the same way as for multiple deletions, we repeated the experiment ten times over both collections and for each value of N_m. The behavior reported for each algorithm, is the average time spent by each algorithm over the ten experiments for each value of N_m.

Figure 9 shows the average behavior of Star and DCS over Reuters (see Fig. 9(a)) and TDT (see Fig. 9(b)) for the selected values of N_m.

As it can be noticed from Fig. 9, DCS outperforms Star when multiple modifications are processed over the Reuters and TDT collections. We conducted other experiments using $N_m = 1500$ and $N_m =$

Table 5
Number of clusters obtained by each algorithm for each document collection

	Collections				
Algorithms	AFP	Reu-Te	Reu-Tr	Reuter	TDT
Star	162	544	727	668	2181
DCS	143	444	502	767	2122
	Collections				
Algorithms	TDT-1	TDT-2	TDT-3	TDT-4	TDT-5
Star	1570	1173	1551	1432	1820
DCS	1414	1208	1501	1123	1226



(a) Reuters, N_m=1000 (b) TDT, N_m=2000

Fig. 9. Behavior of the Star and DCS algorithms over Reuters and TDT for processing N_m multiple modifications.

2000 for Reuters; and $N_m = 3000$ and $N_m = 4000$ for TDT. Using these values for N_m, we observed a behavior similar to that observed in the experiment above described.

4.5. Number of clusters

In this experiment, we compare the number of clusters built by each algorithm, for each document collection, using the β values for which they obtain their best average quality values; i.e., the β values showed in Table 3.

Table 5 shows the number of clusters obtained by each algorithm, for each document collection, using the aforementioned values of β.

As it can be noticed from Table 5, our proposed algorithm obtains, in almost all collections, less clusters than Star.

It is important to mention that we prefer to obtain fewer clusters since the majority of the algorithms, proposed in the state of the art for clustering overlapped collections, build a high number of clusters. If we want to discover the relations or patterns hidden in a collection of objects, we would expect to obtain a number of clusters reasonable smaller than the number of objects in the collection; in this way, we will be able to analyze the resulting clusters. When the number of obtained clusters is very high, as occurs with the majority of the algorithms, to analyze them could become as difficult as analyzing the entire collection; therefore, it loses sense to apply a clustering algorithm over the collection.

5. Conclusions

In this paper, a new dynamic clustering algorithm for overlapping clustering has been proposed. This algorithm, called DCS, builds a set of clusters which could overlap, applying a new graph-covering

strategy for covering the thresholded similarity graph that represents the collection of objects.

From the experiments, we conclude that our proposed algorithm obtains clusterings with a higher quality, considering Fmeasure and Jaccard-index, than those obtained by the Star algorithm. The experiments also showed that the strategy proposed in DCS, 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. It is important to mention that Star needs to know what clusters were affected after each addition/deletion in order to update the clusters; therefore, it can not apply a strategy similar to DCS to process multiple additions/deletions.

Finally, the DCS algorithm obtains less clusters than the Star algorithm. This last characteristic could be of great importance in those applications handling a large set of objects, since it reduces the amount of analysis that must be done over the entire collection.

For all these reasons our algorithm is a better option than the Star algorithm for dynamic environments where the data change frequently.

It is important to mention that, for applying our algorithm in a practical problem, the role of a domain expert is very important. In our algorithm, the tuning of the β parameter could increase or decrease the number of clusters. Therefore, the domain expert would decide, according to his experience and knowledge, what would be a suitable value for this threshold and after that, he can judge the results according to what he expect to obtain with the selected β value.

As future work, we will explore the use of the DCS algorithm in the problem of hierarchical clustering. In this way, we will be able to discover not only what objects are grouped together in a cluster but what relations could exist among the different clusters.

Acknowledgment

This work was partly supported by the National Council of Science and Technology of Mexico under the project CB-2008-01-106443 and grant 32040.

References

[1] R. AbellaPérez and J.E. MedinaPagola, 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, pages 65–72.

[2] C.C. Aggarwal, J. Han, J. Wang and P.S. Yu, A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases – Volume 29*, VLDB '2003, VLDB Endowment, 2003, pages 81–92.

[3] J. Aslam, E. Pelekhov and D. Rus, The star clustering algorithm for static and dynamic information organization. *Journal of Graph Algorithms and Applications* **8**(1) (2004), 95–129.

[4] J. Aslam, K. Pelekhov and D. Rus, Static and dynamic information organization with star clusters. In *Proceedings of the Seventh International Conference on Information and knowledge Management*, 1998, pages 208–217.

[5] J. Aslam, K. Pelekhov and D. Rus, Using star clusters for filtering. In *Proceedings of the Ninth International Conference on Information and Knowledge Management, USA*, 2000, pages 306–313.

[6] A. Banerjee, C. Krumpelman, S. Basu, R. Mooney and J. Ghosh, A new graph-based algorithm for clustering documents. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining* (*KDD2005*), 2005, pages 532–537.

[7] M. Berry, *Survey of Text Mining, Clustering, Classification and Retrieval*, Springer-Verlag, 2004.

[8] G.J. Bloy, Blind camera fingerprinting and image clustering, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **30**(3) (March 2008), 532–534.

[9] L. Chen, L. Tu and Y. Chen, An ant clustering method for a dynamic database. In *Proceedings of the ICMLC 2005*, 2006, pages 169–178.

[10] G. Cselle, K. Albrecht and R. Wattenhofer, Buzztrack: Topic detection and tracking in email. In *Proceedings of the 12th International Conference on Intelligent User Interfaces* (*UI2007*), 2007, pages 446–453.
[11] J. Dunn, Well separated clusters and optimal fuzzy partitions, *Journal of Cybernetics* **4** (1974), 95–104.
[12] H. Elghasel, H. Kheddouci, V. Deslandres and A. Dussauchoy, A partially dynamic clustering algorithm for data insertion and removal, in: V. Corruble, M. Takeda and E. Suzuki, eds, DS 2007, LNAI 4755, 2007, pages 78–90.
[13] M. Ester, H. Kriegel, J. Sander, M. Wimmer and X. Xu, Incremental clustering for mining in a data warehousing environment. In *Proceedings of the 24th Very Large Databases Conference*, 1998, pages 323–333.
[14] R.J. Gil-García, J.M. Badía-Contelles and A. Pons-Porrata, Extended star clustering algorithm. In *Proceedings of the 8th Iberoamerican Congress on Pattern Recognition* (*CIARP2003*), *LNCS 2905*, 2003, pages 480–487.
[15] R.J. Gil-García and A. Pons-Porrata, Hierarchical star clustering algorithm for dynamic document collections. In *Proceedings of the XIII Iberoamerican Congress on Pattern Recognition* (*CIARP2008*), Springer-Verlag Berlin Heidelberg, 2008, pages 187–194.
[16] E. Greengrass, Information retrieval: A survey. Ed Greengrass. DOD Technical Report TR-R52-008-001, 2001.
[17] S. Guha, N. Mishra, R. Motwani and L. O'Callaghan, Clustering data streams. In *Proceedings of the IEEE FOCS Conference*, 2000, pages 359–366.
[18] M. Halkidi and M. Vazirgiannis, Npclu: An approach for clustering spatially extended objects, *Intelligent Data Analysis* **12**(6) (2008), 587–606.
[19] K.M. Hammouda and M.S. Kamel, Efficient phrase-based document indexing for web document clustering, *IEEE Transactions on Knowledge and Data Engineering*, **16**(10) (2004), 1279–1296.
[20] Y. He and L. Chen, A threshold criterion, auto-detection and its use in mst-based clustering, *Intelligent Data Analysis* **9**(3) (2005), 253–271.
[21] A.K. Jain, M.N. Murty and P.J. Flynn, Data clustering: a review, *ACM Computing Surveys* **31**(3) (1999), 264–323.
[22] S. Khy, Y. Ishikawa and H. Kitagawa, Novelty-based incremental document clustering for on-line documents. In *Proceedings of 22nd International Conferencet On Data Enggineering Workshops* (*ICDEW'06*), 2006, page 40.
[23] L. Kuncheva and S. Hadjitodorov, Using diversity in cluster ensembles. In *Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics*, 2004, pages 1214–1219.
[24] A. Lazarevic and Z. Obradovic, Adaptive boosting techniques in heterogeneous and spatial databases, *Intelligent Data Analysis* **5**(4) (2001), 285–308.
[25] Y.G. Liu, X.F. Liao, X.M. Li and Z.F. Wu, A tabu clustering algorithm for intrusion detection, *Intelligent Data Analysis* **8**(4) (2004), 325–344.
[26] P. Mahata, Exploratory consensus of hierarchical clusterings for melanoma and breast cancer. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, March 2008.
[27] R. Nayak, Fast and effective clustering of xml data using structural information, *Knowledge and Information Systems* **14**(2) (2008), 197–215.
[28] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha and R. Motwani, Streaming-data algorithms for high-quality clustering. In *Proceedings of the 18th International Conference on Data Engineering* (*ICDE '02*), 2002, pages 685–709.
[29] M.G.H. Omran, A.P. Engelbrecht and A. Salman, An overview of clustering methods, *Intelligent Data Analysis* **11**(6) (2007), 583–605.
[30] A. Pérez-Suárez and J.E. 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*), *LNAI 4571*, 2007, pages 248–262.
[31] A. Pons-Porrata, R. Berlanga-Llavorí and J. Ruiz-Shulcloper, On-line event and topic detection by using the compact sets clustering algorithm, *Journal of Intelligent and Fuzzy Systems* **12**(3–4) (2002), 185–194.
[32] A. Pons-Porrata, R. Berlanga-Llavorí, J. Ruiz-Shulcloper and J.M. Pérez-Martínez, Jerartop: A new topic detection system. In *Proceedings of the 9th Iberoamerican Congress on Pattern Recognition, LNCS 3287*, Springer-Verlag Berlin Heidelberg, 2004, pages 446–453.
[33] A. Pons-Porrata, J. Ruiz-Shulcloper, R. Berlanga-Llavorí and 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, pages 405–416.
[34] Peter Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, *J Comput Appl Math* **20** (November 1987), 53–65.
[35] G. Salton, A. Wong and C.S. Yang, A vector space model for automatic indexing, *Commun ACM* **18**(11) (1975), 613–620.
[36] G. Sánchez-Díaz and J. Ruiz-Shulcloper, Mid mining: a logical combinatorial pattern recognition approach to clustering in large data sets. In *Proceedings of the 5th Iberoamerican Symposium on Pattern Recognition SIARP 2000*, 2000, pages 475–483.
[37] G. Sánchez-Díaz and J. Ruiz-Shulcloper, A clustering method for very large mixed data sets. In *Proceedings of the First IEEE International Conference on Data Mining* (*ICDM'01*), 2001, page 643.

[38] W. Sia and M. Lazarescu, Clustering large dynamic datasets using exemplar points. In *Proceedings of the 3th International Conference on Machine Learning and Data Mining in Pattern Recognition* (*MLDM 2005*)*, LNAI 3587*, 2005, pages 163–173.
[39] M. Vignes and F. Forbes, Gene clustering via integrated markov models combining individual and pairwise features. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2007, pages 260–270.
[40] O. Zamir and O. Etziony, Web document clustering: A feasibility demonstration. *In Proceedings of the 21st Annual International ACM SIGIR Conference*, 1998, pages 46–54.