RNPS No. 2143
ISSN 2072-6260
Versión Digital

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada
MINISTERIO DE LA INDUSTRIA BÁSICA

REPORTE TÉCNICO
Minería de Datos

SERIE GRIS

Algoritmos jerárquicos y no jerárquicos para la construcción de grupos con traslape en contextos dinámicos

Airel Pérez Suárez, José E. Medina Pagola,
José Fco. Martínez Trinidad y
Jesús A. Carrasco Ochoa

RT_019 **marzo 2013**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

RNPS No. 2143
ISSN 2072-6260
Versión Digital

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada
MINISTERIO DE LA INDUSTRIA BÁSICA

REPORTE TÉCNICO
Minería de Datos

SERIE GRIS

Algoritmos jeráquicos y no jeráquicos para la construcción de grupos con traslape en contextos dinámicos

Airel Pérez Suárez, José E. Medina Pagola,
José Fco. Martínez Trinidad y
Jesús A. Carrasco Ochoa

RT_019 **marzo 2013**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

Tabla de contenido

Lista de figuras

Lista de tablas

Algoritmos jerárquicos y no jerárquicos para la construcción de grupos con traslape en contextos dinámicos

Airel Pérez Suárez[1], José E. Medina Pagola[1], José Fco. Martínez Trinidad[2], y Jesús A. Carrasco Ochoa[2]

[1] Dpto. Minería de Datos, Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV),
La Habana, Cuba
{asuarez,jmedina}@cenatav.co.cu
[2] Coordinación de Ciencias Computacionales, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE),
Puebla, México
{fmartine,ariel}@inaoep.mx

RT_019, Serie Gris, CENATAV
Aceptado: 30 de octubre de 2012

Resumen. El *agrupamiento* es una técnica de la Minería de Datos, que ha sido utilizada en varias áreas. Los algoritmos de agrupamiento, jerárquicos y no jerárquicos, que permiten construir agrupamientos con traslape y que son capaces de actualizar el agrupamiento luego de cambios en la colección, tienen un conjunto de limitaciones que pueden reducir su utilidad en ciertos problemas prácticos. En este trabajo se introducen dos nuevos algoritmos de agrupamiento con traslape, DClustR y DHClustR, que solucionan las limitaciones presentes en los algoritmos del estado-del-arte. Los experimentos realizados sobre colecciones estándares mostraron que los algoritmos propuestos forman agrupamientos con una calidad significativamente superior a los que forman los algoritmos del estado-del-arte. Adicionalmente, estos experimentos mostraron que los algoritmos propuestos son más eficientes en el procesamiento de múltiples cambios en la colección, que los algoritmos relacionados del estado-del-arte.
Palabras clave: minería de datos, algoritmos de agrupamiento con traslape, algoritmos dinámicos.

Abstract. Clustering is a Data Mining technique that has been used in several areas. The hierarchical and non hierarchical clustering algorithms, that address the problem of overlapping clustering, have several limitations that can reduce their usefulness in practical applications. In this work, two new overlapping clustering algorithms, DClustR and DHClustR, which reduce the limitations of the algorithms of the state-of-the-art are introduced. The experiments done over several standard collections showed that the proposed algorithms build clusters having a quality significantly greater, from a statistically point of view, than the one of the clusterings built by the algorithms of the state-of-the-art. Additionally, these experiments showed that the proposed algorithms are more efficient in the processing of multiple changes in the collection, than the algorithms of the state-of-the-art.
Keywords: data mining, overlapping clustering, dynamic algorithms.

1. Introducción

El *agrupamiento* es una de las técnicas fundamentales en el Aprendizaje Automático y en la Minería de Datos [2]. Esta técnica se encarga de organizar una colección de objetos en clases o grupos, de forma tal que los objetos pertenecientes a un mismo grupo sean lo suficientemente similares como para poder inferir que son del mismo tipo y los objetos pertenecientes a grupos distintos sean lo suficientemente diferentes como para poder afirmar que son de tipos diferentes [3].

De forma general, se asume que los objetos a agrupar están descritos por *m* rasgos o variables, cuyos valores pueden ser cuantitativos o cualitativos. Adicionalmente, existen problemas en los que se puede desconocer el valor para algunos de los rasgos [4].

Los algoritmos de agrupamiento pueden ser clasificados atendiendo a diferentes criterios [5]. Entre los criterios más utilizados están: a) si requieren o no conocer el número de grupos a formar a priori, b) el enfoque utilizado para formar los grupos y c) el tipo de los grupos formados.

De acuerdo a si requieren, o no, conocer a priori el número de grupos a obtener, los algoritmos de agrupamiento se clasifican en *restringidos* o *libres*, respectivamente. En la mayoría de los problemas prácticos, el número de grupos a formar es desconocido.

De acuerdo al enfoque utilizado para construir los grupos, los algoritmos se clasifican en *conceptuales* o *clasificatorios*. Los algoritmos conceptuales son aquellos que, además de organizar la colección de objetos en grupos, se encargan de encontrar una descripción intencional de los grupos. Los algoritmos clasificatorios son aquellos que construyen los grupos a partir de las relaciones de semejanzas de los objetos. De forma general, el estudio de métodos de agrupamiento se ha centrado más en los métodos clasificatorios

El conjunto de grupos obtenido por un algoritmo de agrupamiento puede ser *disjunto*, *traslapado* o *difuso*. Los grupos disjuntos son aquellos en los cuales cada objeto pertenece a un solo grupo; *e.g.*, los grupos que se obtienen al agrupar un conjunto de personas por la edad o por el sexo. Por otra parte, los grupos traslapados son aquellos en los cuales los objetos pueden pertenecer a más de un grupo; *e.g.*, los grupos obtenidos al agrupar un conjunto de personas por las enfermedades que padecen. Finalmente, los grupos difusos son aquellos en los que se les asigna a los objetos un valor o un grado de pertenencia a cada uno de los grupos; *e.g.*, los grupos que se obtienen al agrupar un conjunto de personas, de acuerdo al nivel de conocimiento que tengan estas acerca de matemáticas, física, química, política, religión y teología. La mayoría de los algoritmos de agrupamiento reportados en la literatura, construyen grupos disjuntos. Aunque este enfoque ha sido utilizado satisfactoriamente en el aprendizaje no supervisado, existen aplicaciones como la segmentación de textos, la recuperación de información, la detección de comunidades y la bioinformática, entre otras, donde es necesario la construcción de grupos con traslape.

Con base en lo descrito en los párrafos anteriores, este trabajo centrará su atención en los algoritmos de agrupamiento libres, que forman grupos con traslape, siguiendo un enfoque clasificatorio.

Hoy en día, el desarrollo de nuevos algoritmos de agrupamiento continúa siendo objeto de interés debido a su amplia variedad de aplicaciones [6]. A pesar de los resultados que se han alcanzado hasta el momento en el estudio y desarrollo de nuevos algoritmos de agrupamiento con traslape, todavía existen algunas limitaciones en los mismos, que serán tratadas en el marco de este trabajo.

1.1. Descripción del problema

En la literatura se han reportado varios algoritmos para el agrupamiento con traslape. Muchos de estos algoritmos son de propósito general [7,8,9,10,11,12,13,14,15] y otros han sido desarrollados para la detección de comunidades en redes [16,17,18,19,20,21,22,23,24,25,26,27]; no obstante, la mayoría de estos algoritmos no son capaces de procesar, de una forma eficiente, cambios en una colección de objetos ya agrupada.

La capacidad de procesar cambios sobre una colección que se encuentra agrupada, es uno de los criterios utilizados comúnmente para clasificar a los algoritmos de agrupamiento. De acuerdo a este criterio, los algoritmos de agrupamiento pueden ser *estáticos*, *incrementales* o *dinámicos*. Los algoritmos *estáticos* son aquellos que suponen que la colección a agrupar está completamente disponible antes de procesarla. Por tanto, cuando algún objeto es eliminado o adicionado a la colección, estos algoritmos reconstruyen el

agrupamiento desde cero, procesando la colección completa; es decir, los algoritmos estáticos no utilizan los grupos previamente formados para la actualización del agrupamiento. Los algoritmos *incrementales* son aquellos que permiten procesar adiciones a la colección y que actualizan el agrupamiento utilizando los grupos previamente construidos. Por último, los algoritmos *dinámicos* son aquellos que permiten actualizar el agrupamiento cuando la colección cambia producto de adiciones, eliminaciones o modificaciones de objetos. Usualmente, las modificaciones de objetos son procesadas como una eliminación seguida de una adición y de esta forma se tratarán en esta investigación.

La mayoría de los algoritmos para el agrupamiento con traslape que existen son estáticos. Sin embargo, este tipo de algoritmos solo es útil si la colección a procesar no cambiará en un futuro. En otro caso, en aplicaciones donde el conjunto de datos a procesar cambia frecuentemente, los algoritmos estáticos son ineficientes, siendo los algoritmos incrementales y dinámicos los más usados. Algunos ejemplos de este tipo de aplicaciones son la WWW, el análisis del comportamiento de las facturas de una empresa y en el seguimiento de sucesos en flujos de noticias, entre otros.

Los algoritmos de agrupamiento que permiten formar grupos traslapados y que son capaces de procesar cambios en la colección son: Star [28], STC [29], ISC [30], SHC [31], ICSD [32], DCS [33] y DHS [34]. Estos algoritmos utilizan diferentes modelos para representar la colección de objetos y se basan en diferentes conceptos para obtener un conjunto de grupos con traslape. De estos algoritmos, el único capaz de formar jerarquías de grupos es el DHS, que es un algoritmo jerárquico aglomerativo; el resto son algoritmos no jerárquicos. Por otra parte, Star, DCS y DHS son algoritmos dinámicos y el resto son algoritmos incrementales; *i.e.*, solo pueden procesar adiciones de objetos. La complejidad de los algoritmos ISC, SHC, ICSD y DCS es $O(n^2)$. La complejidad de los algoritmos Star y DHS es $O(n^2 \cdot \log^2 n)$ y $O(n^3)$, respectivamente. Por último, la complejidad del algoritmo STC puede llegar a ser exponencial [31].

Una limitación del algoritmo STC está relacionada con su complejidad computacional, la cual puede llegar a ser exponencial. Además, STC fue diseñado específicamente para procesar documentos, lo cual limita su uso en otros contextos.

Los algoritmos Star, ISC, SHC, ICSD y DCS tienen varias limitaciones. Una limitación, como se mostrará en nuestros experimentos, es que construyen grupos con alto traslape. Aunque bien es cierto que formar grupos con traslape es importante y útil para muchas aplicaciones, cuando el nivel de traslape de los grupos es alto se hace muy difícil la obtención de conclusiones útiles. Adicionalmente, existen aplicaciones como la segmentación de documentos por tópicos, en la que un nivel alto de traslape puede significar una mala segmentación [35]. Un ejemplo similar se puede encontrar en el contexto del análisis de redes sociales [20,21].

Otra limitación de estos cinco algoritmos, como se verá en nuestros experimentos, es que construyen una gran cantidad de grupos. Por lo general, en un problema real el número de grupos es desconocido a priori. No obstante, cuando se utiliza un algoritmo de agrupamiento con el objetivo de descubrir relaciones ocultas entre los objetos, se espera que el número de grupos determinado sea pequeño en comparación con el número de objetos. Note que, mientras mayor sea el número de grupos, el análisis de los resultados puede volverse tan difícil como analizar todos los elementos de la colección.

Una limitación de los algoritmos Star y SHC es que no pueden procesar múltiples cambios; es decir, tiene que procesar los cambios uno a uno. Supóngase que el conjunto A contiene los objetos que serán adicionados a la colección. Para procesar estas adiciones, estos dos algoritmos tienen que adicionar los elementos de A uno a uno y actualizar en cada ocasión el agrupamiento. Es importante notar que esta estrategia puede ser computacionalmente costosa pues, si un grupo es afectado por más de un objeto, entonces dicho grupo será actualizado varias veces. Una mejor estrategia sería adicionar todos los elementos y luego actualizar de una sola vez todos los grupos que fueron afectados por las adiciones.

Otra limitación del algoritmo SHC es que necesita ajustar valores para varios parámetros, cuyos valores dependen de la colección a procesar. Por lo general, los usuarios no tienen conocimiento previo acerca de la colección que desean agrupar; por lo tanto, ajustar diferentes parámetros puede ser una tarea muy complicada.

El algoritmo DHS también tiene varias limitaciones. Con el objetivo de acelerar el cálculo de la semejanza entre dos grupos, a través de la medida group-average, DHS utiliza una propiedad demostrada en [36]. No obstante, esta propiedad se puede aplicar solamente cuando los objetos se representan utilizando el modelo VSM [37] y cuando se utiliza a la medida del coseno [38] como función de semejanza. Esta restricción hace que DHS no sea aplicable en problemas donde los objetos no pueden ser representados en este modelo o en aquellos donde la medida del coseno no es la que mejor modela las características del problema.

Otra limitación del algoritmo DHS, como se mostrará en nuestros experimentos, es que construye jerarquías con alto traslape, las cuales están formadas por muchos niveles y muchos grupos. Por último, DHS puede dejar elementos sin agrupar en cada nivel de la jerarquía.

Como se puede concluir de lo expuesto hasta el momento, los algoritmos no jerárquicos para el agrupamiento con traslape, que son capaces de procesar cambios en la colección, tienen varias limitaciones. Estas limitaciones están relacionadas principalmente con: (a) la construcción de un gran número de grupos, (b) la obtención de agrupamientos con un alto nivel de traslape y (c) la necesidad de ajustar varios parámetros, cuyos valores dependen de la colección a agrupar. Por otra parte, la complejidad computacional del algoritmo STC puede llegar a ser exponencial; esto hace a este algoritmo poco útil en aplicaciones reales. Adicionalmente, hay algoritmos como Star y SHC que no pueden procesar múltiples cambios, lo cual dificulta su uso en aplicaciones que procesan colecciones que cambian con mucha frecuencia.

Por último, es importante mencionar que el algoritmo DHS es el único reportado para la construcción de jerarquías traslapadas y el procesamiento de cambios en la colección. No obstante, DHS presenta un conjunto de limitaciones que pueden dificultar su uso en aplicaciones prácticas. Estas limitaciones son: (i) imposición de restricciones al espacio de representación de los objetos a agrupar y a la medida de semejanza entre éstos, (ii) el no agrupamiento de todos los objetos de la colección y (iii) obtención de jerarquías con muchos niveles y muchos grupos. Adicionalmente, la complejidad del algoritmo DHS es $O(n^3)$ y esto puede resultar poco útil en aplicaciones que procesen grandes volúmenes de datos.

En este trabajo se proponen dos algoritmos para el agrupamiento con traslape, que son capaces de procesar eficientemente cambios en una colección. Los algoritmos propuestos se llaman DClustR y DHClustR. El primero, DClustR (del inglés Dynamic Overlapping Clustering based on Relevance), es un algoritmo no jerárquico que introduce una nueva estrategia para construir el agrupamiento, que le permite reducir las limitaciones de los algoritmos no jerárquicos previos. Adicionalmente, el algoritmo propuesto introduce una estrategia que le permite la eficiente actualización del agrupamiento cuando ocurren múltiples cambios y además, tener una complejidad computacional de $O(n^2)$. El segundo algoritmo, DHClustR (del inglés Dynamic Hierarchical Clustering based on Relevance), es un algoritmo jerárquico aglomerativo que construye una jerarquía de grupos traslapados aplicando sucesivamente el algoritmo DClustR. Adicionalmente, DHClustR introduce un nuevo modelo para la representación de los grupos en la jerarquía y una nueva medida para el cálculo de la semejanza entre dos grupos. Las características anteriores le permiten al algoritmo DHClustR reducir las limitaciones presentes en el algoritmo DHS, teniendo una complejidad computacional más baja que la del algoritmo DHS.

1.2. Organización del reporte

El resto de este documento está estructurado como sigue:

En la sección 2 se presentan los conceptos preliminares necesarios para la introducción de los algoritmos propuestos en este trabajo.

En la sección 3, se introduce la estrategia utilizada por el algoritmo DClustR para la construcción de un conjunto de grupos traslapados, así como la estrategia que le permite actualizar el conjunto de grupos, cuando se realizan múltiples adiciones y/o eliminaciones de objetos de la colección. Finalmente, se expone un conjunto de experimentos en los que se compara el algoritmo DClustR contra los algoritmos del estado-del-arte.

En la sección 4, se introduce el modelo que emplea el algoritmo DHClustR para representar a los grupos en la jerarquía y la función que utiliza para calcular la semejanza entre dos grupos, así como la estrategia que utiliza DHClustR para formar una jerarquía de grupos con traslape y la estrategia para actualizarla cuando ocurren cambios. Adicionalmente, se presenta un conjunto de experimentos en los que se les compara contra el algoritmo DHS.

Por último, en la sección 5 se presentan las conclusiones de este trabajo y se describen, tomando como base los resultados expuestos en este documento, algunas direcciones que se pueden seguir como trabajo futuro.

2. Conceptos preliminares

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos, $\beta \in [0,1]$ un parámetro dado y $S(o_i, o_j)$ una función de semejanza entre los objetos de O, que es simétrica.

Un *grafo pesado de* β*-semejanza*, denotado por $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, S\rangle$, es un grafo no dirigido y pesado en el cual $V = O$ y existe una arista $(v,u) \in \widetilde{E}_\beta$ ssi $v \neq u$ y $S(v,u) \geq \beta$; cada arista $(v,u) \in \widetilde{E}_\beta$ está etiquetada con el valor de $S(v,u)$.

Sean $G_1 = \langle V_1, E_1\rangle$ y $G_2 = \langle V_2, E_2\rangle$ dos grafos cualesquiera. Se dice que G_1 es un *sub-grafo* de G_2 ssi se cumple que $V_1 \subseteq V_2$ y $E_1 \subseteq E_2$.

Sea $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, S\rangle$ un grafo pesado de β-semejanza. Un *subgrafo pesado en forma de estrella* (ws-grafo) en $\widetilde{G}_\beta$, denotado por $G^\star = \langle V^\star, E^\star, S\rangle$, es un subgrafo de $\widetilde{G}_\beta$ que tiene un vértice $c \in V^\star$, tal que existe una arista entre c y el resto de los vértices en $V^\star \setminus \{c\}$. El vértice c se llama *centro* y el resto de los vértices se llaman *satélites*. Los vértices aislados de $\widetilde{G}_\beta$ constituyen ws-grafos *degenerados*.

Sea $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, S\rangle$ un grafo pesado de β-semejanza y $W = \{G_1^\star, G_2^\star, \ldots, G_k^\star\}$ un conjunto tal que $G_i^\star = \langle V_i^\star, E_i^\star, S\rangle, i = 1 \ldots k$, es un ws-grafo en $\widetilde{G}_\beta$. El conjunto W es un *cubrimiento* de $\widetilde{G}_\beta$ ssi $V = \bigcup_{i=1}^{k} V_i^\star$. Adicionalmente, se dirá que un ws-grafo $G_i^\star$ *cubre* a un vértice v ssi $v \in V_i^\star$.

Sea $G_v^\star = \langle V_v^\star, E_v^\star, S\rangle$ un ws-grafo no degenerado, que tiene como centro al vértice v. La semejanza intra-grupo de $G_v^\star$, denotada por $Intra\_sim(G_v^\star)$, es el promedio de semejanza que existe entre todos los pares de vértices de $V^\star$:

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

Calcular $Intra\_sim(G_v^\star)$ a través de la ecuación (1) es $O(n^2)$. Por lo tanto, usar esta ecuación para calcular la semejanza intra-grupo de todos los ws-grafos presentes en $\widetilde{G}_\beta$ tomaría un tiempo $O(n^3)$. Dado que se desea que la complejidad de DClustR no sea mayor de $O(n^2)$, se buscará una forma eficiente de aproximar la ecuación (1) para poder usarla en nuestro algoritmo.

De la definición de grafo pesado de β-semejanza, se tiene que todas las aristas (v,u), tal que $S(v,u) < \beta$, no pertenecen a $\widetilde{G}_\beta$. Como cada ws-grafo $G_v^\star$ es un sub-grafo de $\widetilde{G}_\beta$, estas aristas tampoco están en $G_v^\star$. Por lo tanto, si no se tienen en cuenta estas aristas en el cálculo de $Intra\_sim(G_v^\star)$, la ecuación (1) puede

reescribirse como sigue:

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

Aún cuando se evitaron algunos cálculos, la ecuación (2) todavía es $O(n^2)$. Sin embargo, en esta ecuación puede notarse que, para un ws-grafo $G_v^\star$ dado, el valor de $Aprox\_Intra\_sim(G_v^\star)$ solo depende del número de aristas de $G_v^\star$, así como del peso de estas aristas. Tomando en cuenta la definición de ws-grafo, el valor más bajo de $Aprox\_Intra\_sim(G_v^\star)$ se alcanza cuando las aristas en $E_v^\star$ solamente son las que existen entre el centro v y los satélites de $G_v^\star$. Asumiendo lo anterior como el peor escenario posible, la ecuación (2) puede reescribirse de la siguiente forma:

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

Calcular $Aprox\_Intra\_sim(G_v^\star)$ a través de la ecuación (3) es $O(n)$. Luego, se puede utilizar la ecuación (3) para aproximar el valor de la semejanza intra-grupo de cada ws-grafo $G_v^\star$, asumiendo el peor escenario posible, empleando a lo sumo $O(n^2)$. No obstante, si se observa la ecuación (3) se puede notar que su denominador crece más rápido que su numerador; por lo tanto, un cambio pequeño en la cardinalidad de $V^\star$ puede representar un gran decrecimiento en el valor de $Aprox\_Intra\_sim(G_v^\star)$. Esta característica implica que el valor de $Aprox\_Intra\_sim(G_v^\star)$ tendera a cero para valores grandes de $|V_v^\star|$. Lo anterior puede significar un sesgo para ws-grafos con muchos vértices y hace difícil la comparación entre los valores de $Aprox\_Intra\_sim$ de dos ws-grafos con muchos vértices. Con base en lo anterior, se propone aproximar el valor de $Aprox\_Intra\_sim(G_v^\star)$ por el promedio de semejanza de las aristas de $G_v^\star$ que existen entre el centro v y los satélites. De esta forma, la ecuación (3) se puede reescribir como sigue:

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

3. Algoritmo de agrupamiento dinámico con traslape

En esta sección, se introduce el algoritmo DClustR (del inglés Dynamic Overlapping Clustering based on Relevance). La presentación del algoritmo propuesto está dividida en cinco partes. En la subsección 3.1, se describe la estrategia que utiliza DClustR para formar un conjunto de grupos con traslape. En la subsección 3.2, se describe la estrategia que usa DClustR para la actualización del agrupamiento cuando ocurren múltiples cambios en la colección. En la subsección 3.3, se analiza la complejidad computacional del algoritmo DClustR. En la subsección 3.4, se describen un conjunto de resultados experimentales en los que se evalúa al algoritmo DClustR en relación a los algoritmos del estado-del-arte. Finalmente, en la subsección 3.5 se hace una recapitulación de lo presentado en la sección.

3.1. Formación del agrupamiento

DClustR representa la colección de objetos como un grafo pesado de β-semejanza $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, S\rangle$ y construye un conjunto de grupos con traslape utilizando dos etapas: *inicialización* y *perfeccionamiento*. En la etapa de inicialización se construye un conjunto inicial de grupos, a través del cubrimiento de $\widetilde{G}_\beta$ utilizando ws-grafos. Posteriormente, estos grupos iniciales son procesados en la etapa de perfeccionamiento con el objetivo de reducir el número de grupos y el traslape entre ellos. A continuación se describen las etapas de inicialización y perfeccionamiento.

3.1.1. Fase de inicialización

Como se puede observar de sección 2, un ws-grafo está determinado por su centro. Así, el problema de encontrar un conjunto $W = \{G^{\star}_{c_1}, G^{\star}_{c_2}, \ldots, G^{\star}_{c_k}\}$ de ws-grafos, que constituya un cubrimiento de $\widetilde{G}_{\beta}$, puede transformarse en el problema de encontrar el conjunto $X = \{c_1, c_2, \ldots, c_k\}$ de vértices, tal que $c_i \in X$ es el centro del ws-grafo $G^{\star}_{c_i} \in W$, $\forall i = 1 \ldots k$.

Como cada vértice de $\widetilde{G}_{\beta}$ forma un ws-grafo, todos los vértices de $\widetilde{G}_{\beta}$ se deberían analizar para determinar cuáles adicionar a X. Con el objetivo de podar el espacio de búsqueda y tener un criterio de selección, DClustR introduce el concepto de relevancia de un vértice. Para definir la relevancia de un vértice v, primero se introducen los conceptos de *densidad relativa* y *compacidad relativa* de un vértice v.

Una idea para reducir el número de ws-grafos necesarios para cubrir a $\widetilde{G}_{\beta}$, es seleccionar iterativamente los vértices que tengan mayor grado. De esta forma, se trata de maximizar el número de vértices que son adicionados al cubrimiento de $\widetilde{G}_{\beta}$ en cada iteración. No obstante, el número de vértices que un vértice v puede incorporar al cubrimiento de $\widetilde{G}_{\beta}$, siguiendo la estrategia anterior, depende de los vértices seleccionados previamente. Supóngase que los vértices $v_1, v_2, \ldots, v_k$ fueron seleccionados en las primeras k iteraciones. Sea u el vértice que tiene en la iteración $k+1$ el mayor grado, entre los vértices no seleccionados. Si hay $d \leq k$ vértices previamente seleccionados que son adyacentes a u, entonces el vértice u solo adicionaría $|u.Adj| - d$ vértices al cubrimiento de $\widetilde{G}_{\beta}$, en el mejor de los casos. Note que el resto de los adyacentes de u fueron adicionados al cubrimiento de $\widetilde{G}_{\beta}$ en las iteraciones anteriores. Por otra parte, si existiera un vértice z en la iteración $k+1$ tal que:

i) $|z.Adj| < |u.Adj|$.
ii) Hay d_1 vértices seleccionados en las primeras k iteraciones que son adyacentes a z.
iii) $|u.Adj| - d < |z.Adj| - d_1$.

entonces seleccionar al vértice z en la iteración $k+1$ sería una mejor opción que seleccionar al vértice u.

A partir del párrafo anterior se puede concluir que el número de vértices adyacentes a v, que tengan un grado no mayor que v, es un buen estimado del número de vértices que v puede adicionar al cubrimiento de $\widetilde{G}_{\beta}$ si fuera adicionado al conjunto X. Sin embargo, este número está acotado por el número de adyacentes que tiene el vértice v y por lo tanto, puede haber un sesgo con los vértices que tengan muchos vértices adyacentes. Sea v y u dos vértices que podrían adicionar 6 y 4 vértices al cubrimiento de $\widetilde{G}_{\beta}$, respectivamente, si son seleccionados en la iteración $k+1$. Con base en el análisis presentado párrafo anterior, la mejor opción sería seleccionar el vértice v e incluirlo en X. No obstante, si se sabe que $|v.Adj| = 10$ y $|u.Adj| = 5$, entonces u sería una mejor opción, ya que el por ciento de sus vértices adyacentes que puede incluir es mayor que el por ciento que podría incluir v ((4/5 contra 6/10)). Con base en este razonamiento, se introduce el concepto de *densidad relativa* de un vértice.

La *densidad relativa* de un vértice $v \in V$, denotada por $v.densityR$, se calcula de la siguiente forma:

$$v.densityR = \frac{v.density}{|v.Adj|}, \tag{5}$$

donde $v.density$ es el número de vértices adyacentes a v, que tienen un grado no mayor que el grado de v. La ecuación (5) toma valores en el intervalo [0,1] y determina qué parte del total de sus vértices adyacentes podría v incluir en el cubrimiento de $\widetilde{G}_{\beta}$. Mientras mayor sea el valor de $v.densityR$, mejor es el vértice v para el cubrimiento de $\widetilde{G}_{\beta}$.

Existen algunos aspectos de la definición anterior que se deberían resaltar. Primero, un alto valor de $v.densityR$ también significa que el ws-grafo determinado por v tiene un bajo traslape con los ws-grafos seleccionados previamente. Segundo, como esta propiedad está basada en el grado de los vértices,

utilizarla como el único criterio de selección de vértices podría llevar a la selección de ws-grafos con muchos satélites, pero con un bajo promedio de semejanza entre ellos. Para resolver este problema, se introduce el concepto de *compacidad relativa* de un vértice el que, utilizando la semejanza intra-grupo aproximada a través de la ecuación 4 y de forma análoga a como lo hace el concepto de densidad relativa, establece un order de selección entre los vértices de $\widetilde{G}_\beta$.

La *compacidad relativa* de un vértice $v \in V$, denotada por *v.compactnessR*, se calcula como sigue:

$$v.compactnessR = \frac{v.compactness}{|v.Adj|}, \tag{6}$$

donde *v.compactness* es el número de vértices $u \in v.Adj$, tales que $Aprox\_Intra\_sim(G^\star_v) \geq Aprox\_Intra\_sim(G^\star_u)$, siendo $G^\star_v$ y $G^\star_u$ los ws-grafos determinados por *v* and *u*, respectivamente. Tomando en cuenta la semejanza intra-grupo aproximada, en lugar del grado de los vértices como hace el concepto de densidad relativa, la ecuación (6) determina qué parte del total de sus vértices adyacentes podría *v* incluir en el cubrimiento de $\widetilde{G}_\beta$. La compacidad relativa de un vértice *v* toma valores en el intervalo [0,1] y mientras mayor sea su valor, mejor es el vértice *v* para el cubrimiento de $\widetilde{G}_\beta$.

La *relevancia* de un vértice *v*, denotada por *v.relevance*, combina los criterios de densidad relativa y compacidad relativa, a través del promedio de *v.densityR* y *v.compactnessR*. De esta forma, la relevancia toma valores en el intervalo [0,1] y establece un orden de selección entre los vértices, que tiene en cuenta los órdenes determinados por la densidad y compacidad relativas. Un valor elevado de *v.relevance* se corresponde con un alto valor de *v.densityR* y/o *v.compactnessR*; luego, mientras mayor sea el valor de *v.relevance*, mejor será el vértice *v* para el cubrimiento de $\widetilde{G}_\beta$. A partir del comentario anterior, se puede concluir que los vértices de $\widetilde{G}_\beta$ se deben de analizar en orden descendente de su valor de relevancia. Adicionalmente, otra conclusion importante es que no se deben analizar los vértices que tengan $v.relevance = 0$. Con esta definición, además de establecer un orden de selección entre los vértices, se logra podar el espacio de búsqueda.

La estrategia propuesta para construir el cubrimiento de $\widetilde{G}_\beta$ se compone de tres pasos. En el primer paso, todos los vértices son marcados como *satélites* y aquellos un valor de relevancia mayor que cero son adicionados a una lista L; los vértices aislados son incluidos directamente en X. En el segundo paso, se ordena la lista L en orden descendente, de acuerdo al valor de relevancia de los vértices. Posteriormente, se verifican las siguientes condiciones en cada vértice $v \in L$:

a) *v* no está cubierto.
b) *v* está cubierto pero tiene al menos un vértice adyacente no cubierto. Esta condición evita la selección de ws-grafos que tengan todos sus satélites cubiertos por ws-grafos seleccionados previamente.

Cada vértice $v \in L$ que cumpla al menos una de las condiciones anteriores se inserta en X. Cuando todos los vértices de L fueron procesados, cada ws-grafo seleccionado constituye un grupo en el conjunto de grupos iniciales. Estos ws-grafos son procesados posteriormente en la etapa de perfeccionamiento. luster.

3.1.2. Fase de perfeccionamiento

El objetivo de esta etapa es eliminar los ws-grafos seleccionados en la etapa anterior, que sean *menos útiles*. En este contexto, la utilidad de un ws-grafo $G^\star$ se determina en función del número de satélites de $G^\star$ y del número de satélites que $G^\star$ comparte con otros ws-grafos seleccionados.

La estrategia utilizada para construir el conjunto X es *voraz*. Por lo tanto, puede suceder que una vez construido el conjunto X, algunos de sus vértices puedan eliminarse y el conjunto resultante todavía constituya un cubrimiento de $\widetilde{G}_\beta$. Por ejemplo, si existe un vértice $v \in X$ tal que

(i) El ws-grafo determinado por v ($G_v^\star$) comparte todos sus satélites con otros ws-grafos seleccionados.
(ii) El propio vértice v pertenece al menos a otro de los ws-grafos seleccionados.

entonces se puede eliminar a v from X y $\widetilde{G}_\beta$ todavía estaría cubierto por el conjunto de ws-grafos determinados por los vértices restantes en X. Sin embargo, si existe un vértice $u \in X$, tal que u cumple la condición (ii) pero no la condición (i), entonces u no puede ser eliminado de X. Es importante notar que, incluso cuando la mayoría de los satélites de $G_u^\star$ estuvieran cubiertos por otros ws-grafos seleccionados, la eliminación de u del conjunto X dejaría a los satélites que solo pertenecen a $G_u^\star$ como no cubiertos; *i.e.*, fuera del agrupamiento.

Sea $v \in X$ el vértice que determina el ws-grafo $G_v^\star$ en el cubrimiento de $\widetilde{G}_\beta$. Sea $v.Shared$ el conjunto de vértices que $G_v^\star$ comparte con otros ws-grafos y sea $v.Non\_shared$ el conjunto de vértices que solo pertenecen a $G_v^\star$. En este trabajo, se entenderá que un ws-grafo $G_v^\star$ no es útil para el cubrimiento de $\widetilde{G}_\beta$ ssi se cumplen las siguientes condiciones:

(1) Existe al menos otro ws-grafo que cubre a v.
(2) $|v.Shared| > |v.Non\_shared|$.

Para eliminar un ws-grafo no útil $G_v^\star$, se deben incluir los vértices del conjunto $v.Non\_shared$ en algún otro grupo inicial. Dado que $G_v^\star$ cumple la condición (1), los vértices de $v.Non\_shared$ serán adicionados al ws-grafo con mayor número de satélites, de todos los ws-grafos que cubren a v; de esta forma se permite la creación de grupos con muchos elementos. En este contexto, adicionar los vértices de $v.Non\_shared$ a otro ws-grafo $G_u^\star$ significa adicionar dichos vértices a la lista $u.Linked$; esta lista contiene a los vértices que, sin pertenecer directamente al ws-grafo $G_u^\star$, pertenecen al grupo que define dicho ws-grafo.

La estrategia propuesta para eliminar los ws-grafos no útiles consiste en dos pasos. Primero, se marcan los vértices de X como *no-analizados* y se ordenan en orden descendente, de acuerdo al grado. En el segundo paso, se analiza cada vértice $v \in X$ con el objetivo de eliminar los vértices $u \in (X \cap v.Adj)$ que estén marcados como *no-analizados* y que formen ws-grafos no útiles. Los vértices que pertenecen a X y están marcados como *analizados* son aquellos para los que se determinó que formaban ws-grafos útiles en iteraciones previas y, por lo tanto, no necesitan ser analizados de nuevo. Como u es adyacente a v, el ws-grafo $G_u^\star$ ya cumple la condición (1) y, por lo tanto, solo habría que verificar la condición (2). Posteriormente, Si $G_u^\star$ no es útil, u se elimina de X y los vértices de $u.Non\_shared$ se adicionan a la lista $v.Linked$; en otro caso, se marca a u como *analizado* para que no sea analizado por gusto en iteraciones posteriores. Cuando todos los vértices de $v.Adj$ fueron analizados, el vértice v se marca como *semilla*.

Una vez que todos los vértices de X fueron procesados, el conjunto formado por cada vértice en $\widetilde{G}_\beta$, marcado como *semilla*, conjuntamente con los vértices en $v.Adj$ y $v.Linked$, constituye un grupo en el agrupamiento final.

3.2. Actualización del agrupamiento después de los cambios

En esta sección se describe la estrategia que sigue el algoritmo DOCDC para actualizar un agrupamiento, construido siguiendo la estrategia de la sección anterior, cuando la colección de objetos cambia. La idea principal de esta estrategia es determinar las componentes conexas que contienen a los grupos que resultaron afectados por los cambios y, posteriormente, actualizar el cubrimiento de dichas componentes utilizando los ws-grafos de su cubrimiento y otros nuevos que son seleccionados; como consecuencia de lo anterior, el agrupamiento es actualizado.

Sea $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, S \rangle$ el grafo pesado de β-semejanza que representa a una colección O, que fue agrupada según la estrategia descrita en la sección 3.1. Sea S el conjunto de vértices, marcados como semilla, que

determinan el agrupamiento actual de O. Cuando uno o más vértices son adicionados/eliminados de O, puede ocurrir algunas de las siguiente situaciones:

1) Aparecen algunos vértices no cubiertos. Esta situación ocurre cuando: (i) al menos un vértice adicionado no está cubierto o (ii) fueron eliminados todos los vértices que pertenecían a S y que determinaban grupos que cubrían a un vértice en cuestión.
2) La densidad y/o compacidad de algunos vértices cambia y, por consiguiente, aparece al menos un vértice $u \notin S$, tal que u tiene un promedio de densidad-compacidad mayor que: (i) al menos un vértice $z \in (S \cap u.Adj)$, o (ii) al menos un vértice $v \in S$, que cubre vértices en $u.Adj$. Los vértices como u pueden determinar ws-grafos con mejores características (o sea, más satélites y menos solapamiento con otros ws-grafos) que algunos de los ws-grafos que pertenecen al agrupamiento actual de $\widetilde{G}_\beta$.

Para actualizar el agrupamiento cuando ocurre la situación 1), se debe incluir en S un vértice de $v.Adj \cup \{v\}$, para cada vértice v no cubierto. Por otra parte, para actualizar el agrupamiento cuando ocurre situación 2) es necesario un análisis más detallado.

Un vértice v puede cambiar su relevancia cuando cambia *v.densityR* y/o *v.compactnessR*. La densidad relativa de v puede cambiar si cambia *v.density* o $|v.Adj|$. Análogamente, la compacidad relativa de v puede cambiar si cambian *v.compactness* o $|v.Adj|$. El valor de $|v.Adj|$ cambia solo si se adicionan o eliminan vértices de $v.Adj$. El valor de *v.density* y *v.compactness* puede cambiar si cambia el conjunto $v.Adj$ o si existe al menos un vértice $u \in v.Adj$, tal que $u.Adj$ cambió. A partir de este análisis se puede concluir que un vértice pude cambiar su relevancia si uno o más vértices fueron adicionados/eliminados de su *vecindario*. El vecindario de un vértice v está compuesto por los vértices en $v.Adj$ más los vértices adyacentes de cada vértice en $v.Adj$. Dado que se está teniendo en cuenta que puede haber múltiples adiciones/eliminaciones, puede haber un alto solapamiento en los vértices a analizar y por lo tanto, un gasto de tiempo innecesario. Con base en lo anterior, se puede resumir que un vértice puede cambiar su relevancia si pertenece a las componentes conexas que contengan:

- Vértices que fueron adicionado a $\widetilde{G}_\beta$.
- Vértices que eran adyacentes a alguno de los vértices eliminados de $\widetilde{G}_\beta$.

De esta forma, actualizando el agrupamiento de cada una de las componentes conexas anteriormente descritas se puede actualizar el agrupamiento de $\widetilde{G}_\beta$.

Sea $G' = \langle V', E', S \rangle$ una componente conexa cuyo agrupamiento debe ser actualizado. Sea $S' \subseteq S$ el conjunto de vértices marcados como semilla, que determinan el agrupamiento de G'. Para actualizar el agrupamiento de G', se actualizará el cubrimiento de G'. Es importante mencionar que luego de que ocurren cambios en la colección, algunos vértices que formaban ws-grafos no útiles pueden ya dejar de serlo. Por lo tanto, antes de actualizar el cubrimiento de G' se vacía la lista *v.Linked* de cada vértice $v \in X'$; de esta forma, se permite la creación de nuevos ws-grafos. La estrategia que permite actualizar el cubrimiento de G' está compuesta de cinco pasos.

En el primer paso se recalcula la densidad y compacidad de los vértices de G' y se vacía la lista *v.Linked*, de cada vértice $v \in S'$. En el segundo paso, se construye la lista de candidatos L' (ver más abajo); durante la construcción de esta lista se pueden eliminar vértices de S'. En el tercer paso, se procesan los vértices de L' para seleccionar a aquellos que, conjuntamente con los vértices en S', cubren completamente a G'; para seleccionar estos vértices se utiliza la estrategia descrita en la subsección 3.1.1. Posteriormente, en el cuarto paso, se procesa el conjunto S' para eliminar los ws-grafos no útiles; para este propósito, se utiliza la estrategia descrita en la subsección 3.1.2. A continuación se describe cómo se construye la lista L', que es el único paso desconocido de lo descrito en los pasos anteriores.

Sea $V_s \subseteq (V' \setminus S')$ el conjunto de vértices de G' que tienen un valor de relevancia mayor que cero y que no pertenecen a S'. Para formar la lista L', se procesarán los conjuntos S' y V_s.

En el procesamiento de V_s, se verifican las siguientes condiciones en cada vértice $v \in V_s$: (*i*) v no está cubierto, (*ii*) v tiene al menos un vértice adyacente no cubierto, y (*iii*) existe al menos un vértice $u \in v.Adj$, tal que todos los ws-grafos que cubren a u tienen un valor de relevancia menor que el ws-grafo determinado por v. Si v cumple al menos una de estas condiciones, entonces se adiciona v a L'. Adicionalmente, los vértices u como el de la tercera condición son marcados como *activados*. En el procesamiento de S', todos los vértices adyacentes de cada vértice $v \in X'$ son analizados. Durante este análisis, si un vértice $u \in v.Adj$ tiene mayor valor de relevancia que v, entonces se adiciona u a L' y se marca v como *débil*. Cuando todos los vértices en $v.Adj$ son visitados, si v está marcado como débil entonces se elimina de S' a v. Adicionalmente, si v tiene un valor de relevancia mayor que cero, se adiciona v a la lista L'.

Cuando el cubrimiento de las componentes afectadas fue actualizado, el conjunto determinado por cada vértice v marcado como semilla, conjuntamente con los vértices en $v.Adj$ y en $v.Linked$, determina un grupo en el agrupamiento actualizado.

El pseudocódigo del algoritmo DClustR se muestra en el Algoritmo 1. Para actualizar el grafo $\widetilde{G}_\beta$ luego de los cambios en la colección, se utiliza el procedimiento *UpdGraph*. Para actualizar el cubrimiento de una componente conexa G', se utiliza el procedimiento *UpdCovCompt*. Los pseudocódigos de los procedimientos *UpdGraph* y *UpdCovCompt* se muestran en los Algoritmos 2 y 3, respectivamente.

Algoritmo 1: Algoritmo DClustR

Input: $O = \{o_1, o_2, \ldots, o_n\}$ - Colección de objetos, $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, S\rangle$ - Grafo pesado de β-semejanza, β - umbral de semejanza, A - Conjunto de objetos a adicionar, R - Conjunto de objetos a eliminar
Output: O, $\widetilde{G}_\beta$, SC - Conjunto de grupos

```
   // Actualizando G̃β producto de las adiciones/eliminaciones
1  UpdGraph(O,G̃β,β,A,R,M);
2  "Marcar vértices de G̃β como no-procesado";
   // Actualizar el cubrimiento de cada componente conexa afectada por los cambios
3  foreach vértice v ∈ M do
4      if v no está marcado como procesado then
5          "Construir la componente conexa G' = ⟨V',E',S⟩ que contiene a v";
6          if |V'| = 1 then "Marcar v como semilla";
7          else
8              "Recalcular la relevancia de los vértices de G'";
9              "Construir Vs y X'";
10             "Vaciar la lista u.Linked de cada vértice u ∈ X'";
11             "Construir la lista de candidatos L'";
12             UpdCovCompt(G',L',X');
13         end
14         "Marcar cada vértice de G' como procesado";
15     end
16 end
   // Devolviendo el agrupamiento actualizado
17 SC := ∅;
18 foreach vértice v ∈ V do
19     if v está marcado como semilla then SC := SC ∪ {{v} ∪ v.Adj ∪ v.Linked};
20 end
```

Como se puede apreciar en el Algoritmo 1, el algoritmo DClustR asume que existe un grafo pesado de β-semejanza $\widetilde{G}_\beta$ que representa a la colección actual. Si no existiese tal colección (es decir, si $O = \emptyset$) y esta es la primera vez que se agrupará la colección, entonces $\widetilde{G}_\beta$ es un grafo nulo o vacío; de esta forma, DClustR puede procesar también una colección empezando desde cero.

Aunque DClustR está relacionado con los algoritmos Star [28], ICSD [32] y DCS [33], existen algunas diferencias que se resaltan a continuación. Primero que todo, a diferencia de Star, ICSD y DCS, el algoritmo DClustR construye agrupamientos utilizando el concepto de *relevancia* de los vértices. Co-

Algoritmo 2: Procedimiento UpdGraph

Input: $O = \{o_1, o_2, \ldots, o_n\}$ - Colección de objetos, $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, S\rangle$ - Grafo pesado de β-semejanza, β - umbral de semejanza, A - Conjunto de objetos a adicionar, R - Conjunto de objetos a eliminar
Output: O, $\widetilde{G}_\beta$, M - conjunto de vértices

```
1  M := ∅;
   // Procesando las eliminaciones
2  foreach objeto o ∈ R do
3  |  "Eliminar de G̃β al vértice v que representa al objeto o";
4  |  M := M ∪ {x | x ∈ v.Adj ∧ x no representa a ningún objeto en R};
5  end
6  O := O \ R;
   // Procesando las adiciones
7  foreach objeto o ∈ A do
8  |  "Crear vértice v que represente al objeto o y adicionar v a G̃β";
9  |  M := M ∪ {v};
10 end
11 O := O ∪ A;
```

Algoritmo 3: Procedimiento UpdCovCompt

Input: $G' = \langle V', E', S\rangle$ - Componente conexa, L' - Lista de candidatos, X' - Vértices marcados como semilla en G'
Output: G'

```
   // Seleccionando nuevos vértices para completar el cubrimiento de G'
1  "Ordenar L' descendentemente de acuerdo al valor de relevancia de los vértices";
2  foreach vértice v ∈ L' do
3  |  if v cumple las condiciones a) o b) then X' := X' ∪ {v};
4  end
   // Eliminando los ws-grafos no útiles
5  "Ordenar X' descendentemente de acuerdo al grado de los vértices";
6  "Marcar cada vértice de X' como no-analizado";
7  T := ∅;
8  foreach vértice c ∈ X' do
9  |  foreach vértice v ∈ c.Adj do
10 |  |  if v ∈ X' y v está marcado como no-analizado then
11 |  |  |  if el ws-grafo determinado por v no es útil then
12 |  |  |  |  c.Linked := c.Linked ∪ v.Non_shared;
13 |  |  |  |  X' := X' \ {v};
14 |  |  |  end
15 |  |  |  else "Marcar v como analizado";
16 |  |  end
17 |  end
18 |  T := T ∪ {c};
19 end
20 "Marcar cada vértice en T como semilla";
21 "Marcar cada vértice de V' \ T como satélite";
```

mo se mostrará en los experimentos, esta propiedad le permite a DClustR construir agrupamiento con una calidad estadísticamente significativa respecto a los que construye los algoritmos del estado-del-arte, incluidos Star, ICSD y DCS. Adicionalmente, el proceso de perfeccionamiento que realiza DClustR es totalmente diferente a los procesos que realiza ICSD y DCS; como se mostrará en los experimentos, este proceso de perfeccionamiento le permite a DClustR construir agrupamientos con menos grupos y menos traslape, que los construidos por los algoritmos del estado-del-arte.

Finalmente, es necesario mencionar que el algoritmo DClustR depende del orden de análisis de los objetos; es decir, DClustR podría construir diferentes agrupamientos a partir de una misma colección y mismos parámetros, dependiendo del orden en que se analicen los objetos. No obstante, como se men-

ciona en la subsección 3.4, la desviación estándar entre las calidades de estos diferentes agrupamientos es pequeña.

3.3. Análisis de complejidad computacional

Para determinar la complejidad computacional del algoritmo DClustR, se analizarán cada uno de sus pasos. Sea $n_A = |A|$ y $n_R = |R|$ el número de objetos adicionados y eliminados, respectivamente. Sea n_0 y n el número de vértices de $\widetilde{G}_\beta$ antes y después de procesar los cambios, respectivamente.

En el paso 1 se procesan, utilizando el procedimiento *UpdGraph*, las adiciones y/o eliminaciones de objetos. Como se puede observar en el Algoritmo 2, para actualizar el grafo $\widetilde{G}_\beta$ primero se procesan las eliminaciones y luego las adiciones de objetos; de esta forma, se evita el cálculo innecesario de la semejanza entre objetos de A y R. El número de operaciones realizadas para procesar las eliminaciones (pasos 1-5 del procedimiento *UpdGraph*) es $F_1 = \sum_{i=1}^{n_R} 2 \cdot (n_0 - i)$. En el peor de los casos, si todos los objetos de $\widetilde{G}_\beta$ fueran eliminados, entonces el número de operaciones realizadas quedaría de esta forma $F_1 = 2 \cdot \sum_{i=1}^{n_0} (n_0 - i) = 2 \cdot (1 + 2 + \ldots + (n_0 - 1)) = 2 \cdot \frac{n_0 \cdot (n_0 - 1)}{2} = n_0 \cdot (n_0 - 1)$; por lo tanto, F_1 es $O(n^2)$.

Cuando se procesan las adiciones, es necesario calcular la semejanza entre cada vértice adicionado y el resto de vértices de $\widetilde{G}_\beta$. Luego, en el peor de escenario el número de operaciones realizadas en los pasos 6-11 del Algoritmo 2 es $F_2 = n^2$; así, $F_2(n)$ es $O(n^2)$. Es importante notar que, durante el procesamiento de las eliminaciones y adiciones de objetos, se puede actualizar el valor de la semejanza intra-grupo aproximada de cada ws-grafo de $\widetilde{G}_\beta$, utilizando la expresión (4), sin afectar los tiempos F_1 y F_2.

Con base en el análisis anterior, se puede concluir que el número de operaciones realizadas por el Algoritmo 2 es $F_t = F_1 + F_2$. Por la regla se la suma se tiene que F_t es $O(F_1, F_2)$ y por tanto, F_t es $O(n^2)$. Sea $T_1(n)$ el número de operaciones realizadas en el paso 1 del algoritmo DClustR. De acuerdo a la conclusion anterior, $T_1(n)$ es $O(n^2)$. En el paso 2 hay que visitar los vértices de $\widetilde{G}_\beta$. Luego, el número de operaciones realizadas en este paso es $T_2(n) = n$; por lo tanto, $T_2(n)$ es $O(n)$.

Sea $M = \{v_1, v_2, \ldots, v_k\}$ el conjunto de vértices para los cuales son ejecutados los pasos 3-16. Sea $Z = \{G'_1, G'_2, \ldots, G'_k\}$ un conjunto donde $G'_i = \langle V'_i, E'_i \rangle, i = 1 \ldots k$ es la componente conexa inducida por el vértice $v_i \in M$ y $n_i = |V'_i|$ el número de vértices de G'_i; note que $\sum_{i=1}^{k} n_i \leq n$. El número de operaciones realizadas en los pasos 3-16 depende de las operaciones realizadas en los pasos 5-14. El número de operaciones necesarias para formar la componente conexa $G'_i = \langle V'_i, E'_i \rangle$ en el paso 5 es $T_5(n_i) = n_i^2$; luego, $T_5(n_i)$ es $O(n_i^2)$. El número de operaciones realizadas en los pasos 6-13 depende del número de operaciones realizadas en los pasos 8-12.

Para calcular la relevancia de cada vértice $v \in V$, en el paso 8, primero se calculan los valores de *v.densityR* y *v.compactnessR*. A su vez, para calcular *v.densityR* y *v.compactnessR* se deben calcular los valores de *v.density* y *v.compactness* respectivamente. Los valores de *v.density* y *v.compactness* pueden calcularse en un mismo ciclo. Para calcular *v.density* se necesita comparar el grado de v contra el grado de cada vértice en $v.Adj$. Por otra parte, para calcular *v.compactness* se necesita comparar el valor de semejanza intra-grupo aproximada del ws-grafo inducido por v, contra el valor de semejanza intra-grupo aproximada del ws-grafo inducido por cada vértice en $v.Adj$. En el peor de los casos el número de vértices en $v.Adj$ es $|v.Adj| = n$. Por tanto, el número total de operaciones que se realiza en el paso 8 es en el peor caso, $T_8(n_i) = n_i^2$; luego, $T_8(n_i)$ es $O(n_i^2)$.

Para formar los conjuntos V_s y X', hay que visitar los vértices de G'_i y ver cuáles están marcados como *semilla* y cuáles no. Luego, el número de operaciones realizadas en el paso 9 es $T_9(n_i) = n_i$ y por tanto, $T_9(n_i)$ es $O(n_i)$. Note que, en el paso 9 se puede vaciar la lista $c.Linked$ de cada vértice $c \in X'$ sin afectar $T_9(n_i)$. Como se explicó anteriormente, para construir la lista L' hay que visitar los vértices de $v.Adj$ para

cada vértice $v \in V_s$ y los vértices de $c.Adj$ para cada vértice $c \in X'$. Por tanto, el número de operaciones que se realizan en el paso 11 es $T_{11}(n_i) = n_i \cdot (|V_s| + |X'|)$. Dado que $|V_s| + |X'| \leq n_i$ entonces, en el peor de los casos $T_{11}(n_i) = n_i^2$; por tanto, $T_{11}(n_i)$ es $O(n_i^2)$.

Para determinar el número de operaciones realizadas en el paso 12, se determinará el número de operaciones realizadas en el procedimiento UpdCovCompt para una componente $G'_i = \langle V'_i, E'_i \rangle$ cualquiera. El paso 1 de este procedimiento puede realizarse en $O(n_i \cdot \log n_i))$ utilizando el algoritmo *mergesort* [39]; por lo tanto, P_1 es $O(n_i \cdot \log n_i)$. En el proceso en el cual los candidatos de L' son analizados (pasos 2-4), se verifica para cada vértice $v \in L'$ el cumplimiento de las condiciones (a) y (b). Para verificar dichas condiciones es necesario visitar los vértices en $v.Adj$; por lo tanto, el número de operaciones realizadas en este proceso es $P_{2-4} = |L'| \cdot n_i$. Dado que, en el peor de los casos $|L'| = n_i$, el número de operaciones realizadas en los pasos 2-4 es $P_{2-4} = n_i^2$; luego, P_{2-4} es $O(n_i^2)$. Como el grado de un vértice de G'_i es un número entero en el intervalo [1,$n_i - 1$], siguiendo el principio del palomar (*pigeonhole principle*) el paso 5 puede realizarse con un número de operaciones $P_5 = n_i$; luego, P_5 es $O(n_i)$. El número de operaciones realizadas en el paso 6 es $P_6 = |X'|$. En el peor de los casos $|X'| = n_i$ y por consiguiente, P_6 es $O(n_i)$.

Para calcular el número de operaciones realizadas en los pasos 8-19, en lo siguiente nombrado *ciclo-A*, se requiere un análisis más profundo. Sea $|X'| = q_0$ el número de vértices que se seleccionan en los pasos 2-4. En la primera iteración de *ciclo-A* y durante el ciclo de los pasos 9-17, en lo siguiente nombrado *ciclo-B*, cada vértice $v \in \{c.Adj \cap X'\}$ marcado como *no-analizado* es eliminado de X' o marcado como *analizado*, en los pasos 11-15. Por lo tanto, el conjunto de vértices $v \in \{c.Adj \cap X'\}$ que son procesados en el *ciclo-B*, durante la primera iteración de *ciclo-A*, puede ser dividido en dos subconjuntos: M_1 y R_1. El conjunto M_1 contiene a los vértices que fueron marcados como *analizados* y el conjunto R_1 contiene a los vértices que formaban ws-grafos no útiles y que, por consiguiente, fueron eliminados de X'. Adicionalmente, luego de la ejecución de *ciclo-B* en la primera iteración de *ciclo-A*, quedan en X' un conjunto Z_1 de vértices marcados como *no-analizados*.

Del análisis anterior se puede concluir que, luego de la ejecución de ciclo-B en la primera iteración de ciclo-A, $|M_1|$ vértices de X' fueron marcados como *analizados*, $|R_1|$ vértices fueron eliminados de X' por formar ws-grafos no útiles y $|Z_1|$ vértices quedan en X', marcados como *no-analizados*. Note que $|M_1| + |R_1| + |Z_1| = q_0$. Dado que en X' solo quedan $|Z_1|$ vértices marcados como *no-analizados*, cualquier vértice que se procese durante la ejecución de ciclo-B, en la segunda iteración de ciclo-A, pertenece al conjunto Z_1; por lo tanto, $|M_2| + |R_2| + |Z_2| = |Z_1|$. Siguiendo un razonamiento similar, se obtiene que en la tercera iteración $|M_3| + |R_3| + |Z_3| = |Z_2|$. Generalizando, para todas las iteraciones $j > 1$ se cumple que $\left|M_j\right| + \left|R_j\right| + \left|Z_j\right| = \left|Z_{j-1}\right|$.

El número de operaciones realizadas en una iteración j de *ciclo-A* es $F_j = A + B$. A es el número de operaciones realizadas en *ciclo-B* para cada vértice $v \in (c.Adj \cup X')$, marcado como *no-analizados*. B es el resto de las operaciones realizadas en *ciclo-B*. Para determinar cuándo un vértice v forma un ws-grafo no útil, se necesita visitar cada vértice en $v.Adj$. El número de vértices en $c.Adj \cup X'$ que están marcados como *no-analizados*, durante la ejecución de *ciclo-B*, es $\left|M_j\right| + \left|R_j\right|$. Conociendo que en el peor escenario $|v.Adj| = n_i$, se tiene que $A = n_i \cdot (\left|M_j\right| + \left|R_j\right|)$. Dado que para los vértices que no satisfacen las condiciones del paso 10 no se realiza ninguna otra operación, se tiene que $B = n_i - (\left|M_j\right| + \left|R_j\right|)$. Luego, $F_j = n_i \cdot (\left|M_j\right| + \left|R_j\right|) + n_i - (\left|M_j\right| + \left|R_j\right|)$. Si el número de iteraciones que realmente se ejecutaron de *ciclo-A* es $q \leq q_0$, entonces el número de operaciones realizadas en los pasos 8-19 es:

$$P_{8-19} = \sum_{j=1}^{q} F_j = \sum_{j=1}^{q} \left(n_i \cdot (\left|M_j\right| + \left|R_j\right|) + n_i - (\left|M_j\right| + \left|R_j\right|)\right),$$

$$P_{8-19} = \sum_{j=1}^{q} \left(n_i \cdot (\left|M_j\right| + \left|R_j\right|)\right) + \sum_{j=1}^{q} n_i - \sum_{j=1}^{q} \left(\left|M_j\right| + \left|R_j\right|\right),$$

así,

$$P_{8-19} = n_i \cdot \sum_{j=1}^{q} \left(\left|M_j\right| + \left|R_j\right|\right) + n_i \cdot q - \sum_{j=1}^{q} \left(\left|M_j\right| + \left|R_j\right|\right). \tag{7}$$

Como se mencionó anteriormente, se cumple que:

$$\begin{array}{rcl} |M_1| + |R_1| + |Z_1| & = & q_0, \\ |M_2| + |R_2| + |Z_2| & = & |Z_1|, \\ & \vdots & \\ \left|M_q\right| + \left|R_q\right| + \left|Z_q\right| & = & \left|Z_{q-1}\right|. \end{array} \tag{8}$$

Sumando las ecuaciones en (8) se obtiene:

$$\sum_{j=1}^{q} \left(\left|M_j\right| + \left|R_j\right|\right) + \sum_{j=1}^{q} \left|Z_j\right| = q_0 + \sum_{j=1}^{q-1} \left|Z_j\right|,$$

$$\sum_{j=1}^{q} \left(\left|M_j\right| + \left|R_j\right|\right) + \left|Z_q\right| = q_0,$$

luego,

$$\sum_{j=1}^{q} \left(\left|M_j\right| + \left|R_j\right|\right) = q_0 - \left|Z_q\right|. \tag{9}$$

Substituyendo (9) en la ecuación (7), se tiene que:

$$P_{8-19} = n_i \cdot (q_0 - \left|Z_q\right|) + n_i \cdot q - (q_0 - \left|Z_q\right|),$$

$$P_{8-19} = n_i \cdot q_0 - n_i \cdot \left|Z_q\right| + n_i \cdot q - q_0 + \left|Z_q\right|,$$

agrupando términos de igual signo:

$$P_{8-19} = n_i \cdot q_0 + n_i \cdot q + \left|Z_q\right| - (n_i \cdot \left|Z_q\right| + q_0),$$

dado que $(n_i \cdot \left|Z_q\right| + q_0) \geq 0$ y q, q_0 y $\left|Z_q\right|$ son enteros en el intervalo [0,n_i], se tiene que:

$$P_{8-19} \leq n_i \cdot q_0 + n_i \cdot q + \left|Z_q\right| \leq n_i^2 + n_i^2 + n_i,$$

por tanto:

$$P_{8-19} \leq 2 \cdot n_i^2 + n_i. \tag{10}$$

A partir de (10) se puede concluir que P_{8-19} es $O(n_i^2)$. El número de operaciones realizadas en los pasos 20 y 21 es, en el peor de los casos, $P_{20} = n_i$ y $P_{21} = n_i$; luego, P_{20} es $O(n_i)$ y P_{21} es $O(n_i)$. Con base en el análisis anterior, se tiene que el número de operaciones realizadas en el procedimiento UpdCovCompt para una componente G_i' es $P_t = P_1 + P_{2-4} + P_5 + P_6 + P_{8-19} + P_{20} + P_{21}$. Por la regla se la suma se tiene que P_t es $O(P_1, P_{2-4}, P_5, P_6, P_{8-19}, P_{20}, P_{21})$ y por lo tanto, P_t es $O(n_i^2)$. Sea $T_{12}(n_i)$ el número de operaciones realizadas en el paso 12 del algoritmo DClustR. De acuerdo al análisis anterior, se tiene que $T_{12}(n_i)$ es $O(n_i^2)$. En el paso 14 es necesario visitar todos los vértices de la componente G_i' para marcarlos como procesados. Por lo tanto, el número de operaciones realizadas en este paso es $T_{14}(n_i) = n_i$; así, $T_{14}(n_i)$ es $O(n_i)$.

El número de operaciones que se realizan en los pasos 5-14, para actualizar el agrupamiento de la componente G_i' es:

$$T_{5-14}(n_i) = T_5(n_i) + T_8(n_i) + T_9(n_i) + T_{11}(n_i) + T_{12}(n_i) + T_{14}(n_i),$$

$$T_{5-14}(n_i) = n_i^2 + n_i^2 + n_i + n_i^2 + n_i^2 + n_i,$$

$$T_{5-14}(n_i) = 4 \cdot n_i^2 + 2 \cdot n_i,$$

luego, el número de operaciones realizadas para actualizar el agrupamiento de todas las componentes (pasos 3-16) es:

$$T_{3-16}(n) = \sum_{i=1}^{k} T_{5-14}(n_i) = \sum_{i=1}^{k} \left(4 \cdot n_i^2 + 2 \cdot n_i\right),$$

$$T_{3-16}(n) = 4 \cdot \sum_{i=1}^{k} n_i^2 + 2 \cdot \sum_{i=1}^{k} n_i,$$

dado que $\sum_{i=1}^{k} n_i \leq n$ y que $\sum_{i=1}^{k} n_i^2 \leq \left(\sum_{i=1}^{k} n_i\right)^2$ entonces:

$$T_{3-16}(n) \leq 4 \cdot \sum_{i=1}^{k} n_i^2 + 2 \cdot n \leq 4 \cdot \left(\sum_{i=1}^{k} n_i\right)^2 + 2 \cdot n,$$

luego:

$$T_{3-16}(n) \leq 4 \cdot n^2 + 2 \cdot n. \tag{11}$$

A partir de la expresión (11), se puede concluir que $T_{3-16}(n)$ es $O(n^2)$. Para devolver el agrupamiento final en los pasos 17-20, se visitan los vértices en $v.Adj$ y en $v.Linked$, de cada vértice $v \in V$ que esté marcado como *semilla*; para los vértices no marcados como *semilla* no se realiza ninguna operación. Luego, el número de operaciones realizadas en los pasos 17-20 es:

$$T_{17-20}(n) = \sum_{i=1}^{k} |v_i.Adj| + |v_i.Linked| + n - k,$$

donde k es el número de vértices marcados como *semilla* en $\widetilde{G}_\beta$. Para cualquier vértice v marcado como *semilla*, se cumple que $v.Adj \cap v.Linked = \emptyset$. Luego, $|v_i.Adj| + |v_i.Linked| \leq n$ y por lo tanto:

$$T_{17-20}(n) \leq \sum_{i=1}^{k} n + n - k,$$

como en el caso peor el número de vértices marcados como *semilla* es n, se tiene que:

$$T_{17-20}(n) \leq n^2. \tag{12}$$

A partir de (12) se puede concluir que T_{17-20} es $O(n^2)$. Finalmente, el número total de operaciones realizadas en el algoritmo DClustR es $T_{1-20}(n) = T_1(n) + T_2(n) + T_{3-16}(n) + T_{17-20}(n)$. Aplicando la regla de la suma, se tiene que $T_{1-20}(n)$ es $O(T_1(n), T_2(n), T_{3-16}(n), T_{17-20}(n))$; por lo tanto, $T_{1-20}(n)$ es $O(n^2)$ y la complejidad computacional del algoritmo DClustR es $O(n^2)$.

3.4. Resultados experimentales

En esta sección, se presentan los resultados de una serie de experimentos en los que se evalúa al algoritmo DClustR de acuerdo a varios criterios.

Los primeros tres experimentos estuvieron enfocados en evaluar el algoritmo DClustR de acuerdo a la calidad de los grupos formados, el número de grupos construidos y el traslape de dichos grupos. En estos tres experimentos, se compararon los resultados obtenidos por DClustR usando los tres criterios de evaluación antes mencionados, contra los resultados obtenidos por los algoritmos no jerárquicos del estado-del-arte: Star [28], ISC [30], SHC [31], ICSD [32] y DCS [33]. Adicionalmente, en esta lista se incluyeron los algoritmos Estar [13], Gstar [15] y ACONS [12], que son algoritmos estáticos que han obtenidos buenos resultados en la construcción de grupos con traslape.

El cuarto experimento se centró en evaluar el algoritmo DClustR de acuerdo al tiempo empleado para procesar múltiples adiciones y/o eliminaciones de objetos. En los experimentos con múltiples adiciones, se contrastaron los resultados obtenidos por DClustR con los resultados obtenidos por los algoritmos Star, ISC, SHC, ICSD y DCS; estos algoritmos de agrupamiento son los que permiten formar grupos traslapados y procesar adiciones de objetos. En los experimentos con múltiples eliminaciones y múltiples modificaciones, se compararon los resultados de DClustR con los resultados de Star y DCS; estos son los únicos algoritmos dinámicos no jerárquicos, reportados en la literatura, que permiten formar grupos con traslape.

Todos los algoritmos usados en los experimentos fueron implementados en C++ y compilados utilizando el compilador G++. Los experimentos se realizaron en una computadora con un procesador Intel Core 2 Duo a 1.86 GHz, 2 GB de memoria RAM y con sistema operativo RedHat Enterprise Linux 5.3.

3.4.1. Colecciones de prueba

Dado que en esta investigación doctoral se analiza el problema del agrupamiento con traslape, los algoritmos será evaluados en la tarea del agrupamiento de documentos, en donde es común que un documento pertenezca a más de un tópico o clase.

Las colecciones de documentos utilizadas en los experimentos fueron construidas a partir de cinco colecciones estándares, o *corpus*, utilizadas en el agrupamiento de documentos: AFP, Reuters-21578, TDT2, CISI y CACM; cada corpus se distribuye con su *ground-truth*. El corpus AFP fue descargado del sitio http://trec.nist.gov y contiene noticias en español que fueran publicadas por la agencia de noticias AFP durante el año 1994. Reuters-21578 se obtuvo del sitio http://kdd.ics.uci.edu y contiene noticias en inglés publicadas por la agencia de noticias Reuters durante el año 1987. El corpus TDT2 se puede obtener del sitio http://www.nist.gov/speech/tests/tdt.html y contiene noticias en inglés que provienen de diferentes agencias periodísticas y que fueran publicadas en el año 1998, durante el período de Enero a Junio. Por otra parte, los corpus CISI y CACM contienen abstracts de artículos científicos, que fueran utilizados en el proyecto SMART en tareas de recuperación de información. Estas dos colecciones pueden descargarse del sitio ftp://ftp.cs.cornell.edu/pub/smart. De cada corpus se eliminaron aquellos documentos que no tenían asociado tópicos en el *ground-truth*.

A partir de los corpus descritos anteriormente, se construyeron doce colecciones de prueba: (1) la colección AFP fue construida utilizando todas las noticias del corpus AFP, (2) la colección Reu-Te se construyó utilizando las noticias del corpus Reuters que tenían la etiqueta “Test”, (3) la colección Reu-Tr se construyó utilizando las noticias del corpus Reuters que tenían la etiqueta “Train”, (4) la colección Reuters, es la unión de las colecciones Reu-Te y Reu-Tr, (5) la colección cisi se construyó utilizando los documentos del corpus CISI, (6) la colección cacm se construyó utilizando los documentos del corpus CACM, (7) la colección TDT se construyó usando todas las noticias del corpus TDT2. Por último, a partir

de la colección TDT se construyeron cinco sub-colecciones llamadas TDT-1, TDT-2, TDT-3, TDT-4 y TDT-5. Para construir estas cinco sub-colecciones, las noticias de TDT fueron separadas aleatoriamente en 5 conjuntos y posteriormente, cada sub-colección se formó con las noticias contenidas en tres conjuntos seleccionados aleatoriamente.

Las características de las doce colecciones de prueba descritas anteriormente se muestran en la Tabla 1. Las columnas etiquetadas como "#Documentos" y "#Términos" representan el número de documentos y términos contenidos en cada colección. La columna etiquetada como "#Clases" representa el número de clases en las que los expertos organizaron el *ground-truth* de cada colección. La columna etiquetada como "Traslape" representa el promedio de clases en las que está incluido cada documento de la colección [34].

Tabla 1. Características de las colecciones de prueba que se utilizarán en los experimentos.

Nombre	#Documentos	#Términos	#Clases	Traslape
AFP	695	11785	25	1.023
Reu-Te	3587	15113	100	1.295
Reu-Tr	7780	21901	115	1.241
Reuter	11367	27083	120	1.258
TDT	16006	68019	193	1.188
TDT-1	8602	51764	176	1.202
TDT-2	7404	44610	178	1.173
TDT-3	10258	53706	174	1.189
TDT-4	10074	53036	172	1.182
TDT-5	11328	55923	182	1.180
cacm	433	3038	52	1.499
cisi	1162	6976	76	2.680

En los experimentos, los documentos fueron representados utilizando el Modelo de Espacio Vectorial o (VSM, por sus siglas en inglés) [37]. Los términos de los documentos representan los lexemas de las palabras que ocurren al menos una vez en toda la colección. Los lexemas fueron extraídos de los documentos utilizando la herramienta Tree-tagger; éste software puede descargarse del sitio http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger. Las palabras vacías tales como: artículos, preposiciones, adverbios, entre otras, fueron eliminadas de los documentos.

Los términos de los documentos fueron estadísticamente pesados utilizando el esquema de la frecuencia del término normalizada por la frecuencia máxima del documento. La frecuencia máxima del documento es la frecuencia del término que más aparece dentro del documento [40]. Para determinar la semejanza entre dos documentos se utilizó la medida del coseno [38].

3.4.2. Medidas de evaluación

Existen tres tipos de medidas de evaluación de agrupamientos: *externas*, *internas* y las *relativas* [3]; de estos tres tipos de medidas, las más usadas son las externas [1]. Las medidas externas evalúan un algoritmo de agrupamiento, utilizando un conjunto de clases, comúnmente llamado *ground-truth*, construido manualmente por expertos. Mientras más semejante sean el conjunto de grupos y el *ground-truth*, mejor será el algoritmo de agrupamiento.

En la literatura se han propuestos varias medidas externas para evaluar la eficacia de los algoritmos de agrupamiento: Purity and Inverse Purity [41], F1-measure [42], Jaccard coefficient [43], FM index [44], Entropy [45], Class Entropy [46], Fmeasure [7] y V-measure [47], entre otras. Estas medidas difieren en cuanto a sus bases matemáticas, sus sesgos y sus limitaciones. Existen trabajos que han utilizados medidas basadas en *conteo de pares*, como la Fmeasure y el Jaccard coefficient, para evaluar agrupamientos con traslape [7,15,12,32,33]. Existen también otras trabajos han usado medidas basadas en *cotejo de conjuntos*, como la F1-measure, para evaluar agrupamientos con traslape [13,34]. No obstante, ninguna de estas

medidas utilizadas están definidas, al menos explícitamente, para evaluar agrupamiento con traslape; es decir, estas medidas no permiten evaluar el hecho de que, en un agrupamiento con traslape ideal, los objetos que compartan *n* clases en el *ground-truth* deberían compartir *n* grupos en el agrupamiento.

Según nuestro conocimiento, en la literatura solo existen dos trabajos que proponen y analizan medidas para la evaluación de agrupamientos con traslape [1,48].

En [1], Amigo *et al.* proponen una nueva medida externa para evaluar agrupamientos con traslape. Esta medida se llama FBcubed y se calcula utilizando variaciones de las medidas Bcubed precision y recall, propuestas en [49].

Sea $D(o)$ el conjunto de objetos que, incluyendo a *o*, comparten al menos un grupo con el objeto *o*. La medida Bcubed precision del objeto *o*, que propone Amigo *et al.*, se denota por $Bcubed_{pre}(o)$ y se calcula de la siguiente forma:

$$Bcubed_{pre}(o) = \frac{\sum_{e \in D(o)} MBcubed_{pre}(o,e)}{|D(o)|}, \tag{13}$$

donde $MBcubed_{pre}(o,e)$ es la Bcubed precision del objeto *o* respecto al objeto *e* y se calcula como sigue:

$$MBcubed_{pre}(o,e) = \frac{MIN(|C(o) \cap C(e)|, |L(o) \cap L(e)|)}{|C(o) \cap C(e)|},$$

donde $C(o)$ y $L(o)$ son los grupos y clases a las que el objeto *o* pertenece; $C(e)$ y $L(e)$ están definidas de la misma forma que $C(o)$ y $L(o)$.

Sea $H(o)$ el conjunto de objetos que, incluyendo a *o*, comparten al menos una clase con el objeto *o*. La medida Bcubed recall del objeto *o*, que propone Amigo *et al.*, se denota por $Bcubed_{rec}(o)$ y se calcula de la siguiente forma:

$$Bcubed_{rec}(o) = \frac{\sum_{e \in H(o)} MBcubed_{rec}(o,e)}{|H(o)|}, \tag{14}$$

donde $MBcubed_{rec}(o,e)$ es la Bcubed recall del objeto *o* respecto al objeto *e* y la misma se calcula como sigue:

$$MBcubed_{rec}(o,e) = \frac{MIN(|C(o) \cap C(e)|, |L(o) \cap L(e)|)}{|L(o) \cap L(e)|}.$$

La Bcubed precision total se denota por $Bcubed_{pre}$ y se calcula como el promedio de Bcubed precision de cada uno de los objetos en la colección. La Bcubed recall total, denotada por $Bcubed_{rec}$, calcula como el promedio de Bcubed recall de cada uno de los objetos en la colección. Finalmente, la medida FBcubed se calcula como la media harmónica de $Bcubed_{pre}$ y $Bcubed_{rec}$, es decir:

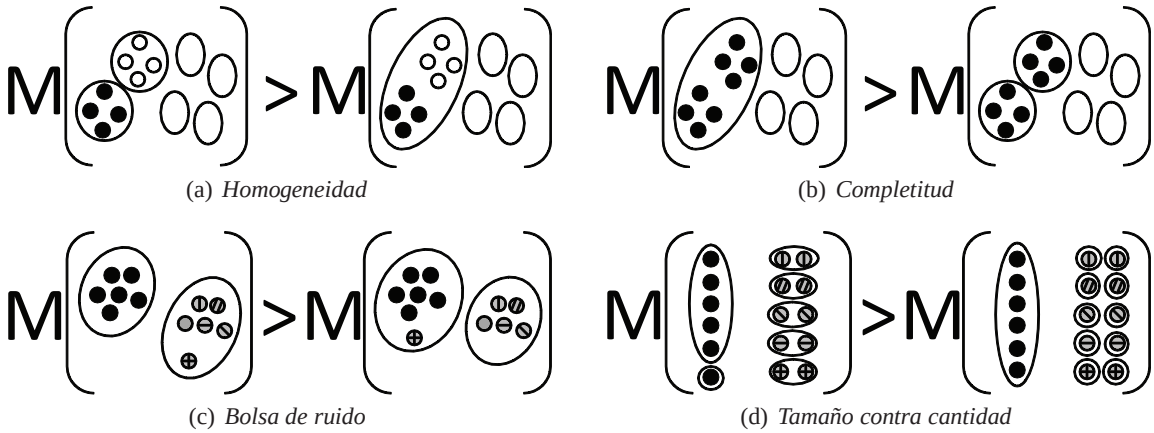
$$FBcubed = \frac{2 \cdot Bcubed_{pre} \cdot Bcubed_{rec}}{Bcubed_{pre} + Bcubed_{rec}}. \tag{15}$$

Es importante notar que la forma en la cual la Bcubed precision y recall de un objeto son definidas, permiten a la medida FBcubed detectar situaciones en las que:

1) El algoritmo de agrupamiento no es capaz de capturar completamente la relación entre dos objetos. Esta situación pasa cuando existen al menos dos objetos que comparten más clases que grupos. En este caso, el valor de la Bcubed recall total decrecerá y consecuentemente, el valor de la medida FBcubed también decrecerá.
2) El algoritmo de agrupamiento introduce más información de lo necesario. Esta situación pasa cuando existen al menos dos objetos compartiendo más grupos que clases. En este caso, el valor de la Bcubed precision total decrecerá y consecuentemente, el valor de la medida FBcubed también decrecerá.

Adicionalmente, La medida FBcubed satisface cuatro requerimientos que le permite evaluar varias características deseables en un agrupamiento [1]. Estos cuatro requerimientos son:

a) *Homogeneidad*. Esta restricción establece que un mismo grupo no debe contener objetos de diferentes clases (ver Figura 3(a)).
b) *Completitud*. Esta restricción establece que los objetos pertenecientes a una misma clase deben pertenecer a un mismo grupo (ver Figura 3(b)).
c) *Bolsa de ruido*. Esta restricción establece que es preferible obtener un agrupamiento que tenga grupos homogéneos y un grupo ruidoso, a obtener un agrupamiento en el cual todos los grupos no sean homogéneos (ver Figura 1(c)).
d) *Tamaño contra cantidad*. Esta restricción establece que un error pequeño en un grupo grande es mejor que varios errores en grupos pequeños; o dicho de otra forma: separar un objeto de su clase de tamaño $n > 2$ es mejor que separar n clases binarias (ver Figura 1(d)).



(a) *Homogeneidad* (b) *Completitud*

(c) *Bolsa de ruido* (d) *Tamaño contra cantidad*

Fig. 1. Ilustrando los cuatro requerimientos que satisface la medida FBcubed [1].

Como se puede observar de la Figura 1, estos requerimientos son intuitivos y expresan características importantes que debieran ser evaluadas por una medida externa M cualquiera. Aún más, en [1] se mostró que ninguna de las medidas externas más usadas para evaluar agrupamientos, satisface los cuatro requerimientos.

En [48], Ramírez *et al.* proponen tres medidas externas para la evaluación de agrupamientos con traslape: GFM, PCMP, y Clustering Recall (R_U). La primera, GFM, es una generalización de la medida Fowlkes-Mallows index (FM index) [44], la cual es una medida basada en conteo de pares. Por otra parte, PCMP (de Partial Class Match Precision), es también una medida basada en conteo de pares que, a diferencia de GFM, mide la probabilidad de seleccionar aleatoriamente dos objetos de la misma clase, a partir de una muestra aleatoria de un grupo. Como se planteó en [48], las dos medidas anteriores trabajan bajo el supuesto de que el traslape está restringido a pares de clases; esta restricción no se cumple necesariamente en problemas reales. Además, como se mostró en [1], las medidas basadas en conteo de pares no satisfacen los requerimientos *c* y *d*. La medida R_U se basa en el concepto de *recall*, tomado del área de Recuperación de Información, y mide el promedio de objetos de cada clase que fueron incluidos en todos los grupos. R_U es una medida basada en cotejo de conjuntos que, como purity [41], no satisface los requerimientos *b* y *c*. Aún más, ninguna de estas tres medidas propuestas en [48], son capaces de evaluar el hecho de que, en un agrupamiento con traslape ideal, los objetos que compartan n clases en el *ground-truth* deberían compartir n grupos en el agrupamiento.

Con base en lo expuesto en esta subsección y con el objetivo de realizar una comparación justa entre los algoritmos, se utilizará la medida FBcubed para evaluar la eficacia de los agrupamientos formados por cada algoritmo. Una explicación más detallada acerca de la medida FBcubed, conjuntamente con un caso de estudio, puede encontrarse en [1].

3.4.3. Calidad de los grupos

En este experimento, se comparan los algoritmos de acuerdo a la calidad de los grupos que forman para cada colección de documentos. Para determinar la calidad de los grupos formados por cada algoritmo se utiliza la medida FBcubed [1]. Para una mejor comprensión de cómo se llevó a cabo este experimento, a continuación se describe el procedimiento que se siguió con la colección AFP.

Primeramente, todos los algoritmos se ejecutaron utilizando valores de β en el intervalo [0.10,0.50], con un incremento de 0.01; es decir, los valores de β utilizados fueron β=0.10,0.11, …, β=0.50. En el caso del algoritmo SHC, se probaron además diferentes valores en el intervalo [0,0.10] para el parámetro ε y valores en el intervalo [0,β-0.05] para el parámetro HR_{min}; se seleccionaron aquellos valores de ε y HR_{min} con los que se obtuvo los mejores resultados. A continuación, se determinó el valor de FBcubed que obtiene cada algoritmo para los diferentes valores de β. Como los algoritmos ISC y Estar no dependen del orden de los datos, se seleccionó su valor más alto de FBcubed, para los diferentes valores de β, como su mejor desempeño sobre AFP. Para Star, SHC, Gstar, ACONS, ICSD, DCS y DClustR, que dependen del orden de análisis de los objetos, se repitió el experimento anterior veinte veces, variando cada vez los órdenes de los documentos. Para cada uno de estos algoritmos, el promedio más alto de FBcubed, para cada uno de los valores de β, fue seleccionado como su mejor desempeño sobre AFP.

Para el resto de las colecciones se utilizó el procedimiento anteriormente descrito. Durante estas evaluaciones se observó lo siguiente

- Para valores de β mayores que 0.50 o menores de 0.10, la calidad de los agrupamientos construidos por todos los algoritmos decreció. Por esta razón no se utilizaron valores de β fuera del intervalo mencionado anteriormente.
- Incluso cuando los algoritmos Star, SHC, Gstar, ACONS, ICSD, DCS y OClustR dependen del orden de análisis de los objetos, la desviación estándar de los valores de FBcubed de los agrupamientos construidos para los diferentes órdenes de los documentos, fue menor que 0.01, para cada valor de β. Esto significa que el valor de FBcubed de estos algoritmos varió poco para los diferentes órdenes; por lo tanto, se puede utilizar el valor promedio como su mejor desempeño para cada valor de β.

En la Tabla 6 se muestra la mejor evaluación que cada algoritmo logra, respecto a la medida FBcubed, sobre las colecciones de prueba.

Como puede verse en la Tabla 6, DClustR obtiene mejores valores de calidad, de acuerdo a la medida FBcubed, que el resto de los algoritmos. Para resumir los resultados mostrados en esta tabla, se determinó la significancia estadística de los resultados obtenidos por DClustR, en comparación con los obtenidos por el resto de los algoritmos. Para este cálculo se utilizó el test de Mann-Whitney con un 95% de certeza. Una información detallada acerca de este test, así como una implementación del test pueden encontrarse en el sitio http://faculty.vassar.edu/lowry/webtext.html. El resultado de este test mostró que DClustR construye agrupamientos con una calidad estadísticamente significativa respecto a los construidos por el resto de los algoritmos.

Tabla 2. Mejores evaluaciones de cada algoritmo, respecto a la medida FBcubed, en las colecciones de prueba. En cada colección, los valores más altos de calidad aparecen en negrita.

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

3.4.4. Número de grupos

En esta sección, se compara el número de grupos que construye cada algoritmo sobre cada colección de prueba, cuando alcanza su mejor evaluación sobre dicha colección (ver Tabla 6). En la Tabla 3 se muestra el resultado de este experimento.

Tabla 3. Número de grupos que forma cada algoritmo sobre cada colección. Los valores más bajos aparecen en negrita.

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

Como se muestra en la Tabla 3, DClustR construye en casi todas las colecciones, menos grupos que el resto de los algoritmos. Es importante resaltar que DClustR obtiene estos resultados sin afectar la calidad de los grupos (ver Tabla 6); de esta forma, nuestro algoritmo produce agrupamientos de alta calidad, que pueden resultar más fáciles de analizar que aquellos agrupamientos construidos por el resto de los algoritmos usados en este experimento.

3.4.5. Traslape entre los grupos

En este experimento, se compara el traslape del agrupamiento construido por cada algoritmo para cada una de las colecciones, cuando se utilizan los parámetros con los cuales cada algoritmo obtiene los mejores valores de FBcubed (ver sección 3.4.3). El traslape del agrupamiento construido por un algoritmo es el promedio de grupos en los cuales está incluido cada objeto de la colección [34]. Si este promedio es mayor que 1, entonces el algoritmo permite construir grupos con traslape. En la Tabla 4, se muestra el traslape del conjunto de grupos que construye cada algoritmo para las colecciones de prueba.

Tabla 4. Traslape del agrupamiento que cada algoritmo construye para las colecciones de prueba. Los valores más bajos por colección aparecen en negrita.

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

Como puede verse en la Tabla 4, DClustR construye agrupamientos que tienen un traslape menor que el resto de los algoritmos. Es importante resaltar que estos resultados de DClustR se obtienen sin afectar la calidad de los grupos formados. Esta característica puede ser importante para aplicaciones en las que se requiera que exista un bajo traslape entre los grupos, como por ejemplo: la segmentación de documentos por tópicos [35], la organización de documentos web [29,31], entre otras.

Adicionalmente, se llevó acabo otro experimento que se enfocó en determinar, para cada colección de prueba, qué tan lejano está el traslape del agrupamiento obtenido por cada algoritmo, del traslape del *ground-truth* de cada colección.

Para cada colección, se calculó el cociente entre el traslape del agrupamiento construido por el algoritmo y el traslape del *ground-truth* de dicha colección; de aquí en adelante, se referirá a este cociente con el nombre de *traslape relativo*. Si el traslape relativo es mayor que 1 entonces, el algoritmo produce más traslape que el existente en la colección. En otro caso, si el traslape relativo es menor a 1 entonces, el algoritmo produce menos traslape que el que tiene la colección. Un traslape relativo igual a 1 significa que ambos, el agrupamiento obtenido por el algoritmo y el *ground-truth* de la colección, tienen el mismo traslape. Por lo tanto, el valor absoluto de $(TR-1)$ dará una idea de cuán lejano está el traslape obtenido por el algoritmo, del traslape de la colección. Mientras más cercano a cero, mejor será el algoritmo.

En la Tabla 5 se muestra, para cada colección de prueba, cuán lejos está el traslape del agrupamiento que construye cada algoritmo, del traslape existente en la colección.

Como puede observarse en la Tabla 5, el algoritmo DClustR es el de mejor desempeño en casi todas las colecciones de prueba. La colección cisi tiene un traslape relativamente alto en comparación con el resto de las colecciones de prueba. Por lo tanto, es de esperar que el algoritmo ISC que construye grupos con mucho traslape, obtenga resultados cercanos al traslape del *ground-truth*. Por último, es importante notar que DClustR tiene, en el caso promedio, un comportamiento al menos dos veces mejor que el segundo mejor algoritmo (ISC).

3.4.6. Eficiencia en el procesamiento de cambios

En esta sección, se comparan los algoritmos de acuerdo al tiempo que tardan procesando múltiples adiciones, eliminaciones, así como modificaciones sobre la colección TDT. En los experimentos de adición, se compara el tiempo empleado por los algoritmos Star, ICSD, ISC, SHC, DCS y DClustR. Por ora parte, en los experimentos de eliminación y modificación se comparan los tiempos empleados por los algoritmos Star, DCS y DClustR, que son los únicos capaces de procesar estos cambios.

Tabla 5. Traslape producido por cada algoritmo en relación al traslape existente en la colección. Los valores más bajos por colección aparecen en negrita.

	Algoritmos								
Col.	Star	ISC	Estar	Gstar	ACONS	ICSD	SHC	DCS	DClustR
AFP	0.67	0.61	1.47	1.26	1.43	1.47	1.38	1.47	**0.15**
Reu-Te	1.63	0.38	4.71	4.20	4.14	4.90	9.14	4.90	**0.08**
Reu-Tr	3.46	0.48	8.78	9.54	9.20	9.73	22.63	9.73	**0.26**
Reuter	3.34	0.44	11.65	11.29	11.30	14.31	36.80	14.31	**0.22**
TDT	3.05	0.58	49.01	57.45	54.88	58.74	66.97	58.74	**0.26**
TDT-1	1.81	0.53	35.79	39.83	37.60	40.29	38.86	40.29	**0.19**
TDT-2	1.88	0.53	28.93	31.27	30.94	31.27	32.95	31.27	**0.19**
TDT-3	2.21	0.55	34.66	37.75	36.70	37.87	44.24	37.87	**0.28**
TDT-4	2.45	0.55	35.67	39.26	38.12	40.22	46.41	40.22	**0.23**
TDT-5	2.38	0.59	36.31	42.61	40.05	43.26	49.67	43.26	**0.23**
cacm	0.54	0.21	1.31	1.13	1.13	0.81	0.33	0.81	**0.16**
cisi	0.54	**0.21**	1.93	1.79	1.90	1.81	1.60	1.81	0.41
Prom.	1.99	0.47	20.85	23.12	22.28	23.72	29.25	23.72	**0.22**

En la Figura 5, se muestra el comportamiento de los algoritmos Star, ICSD, ISC, SHC, DCS y DClustR, en el procesamiento de múltiples adiciones sobre la colección TDT. En esta figura, cada curva representa el tiempo promedio que emplea cada algoritmo para agrupar subcolecciones de tamaño 2000, 4000, 6000, …, 16 000; este promedio se calculó sobre veinte ejecuciones de los algoritmos sobre TDT, variando el orden de los documentos.

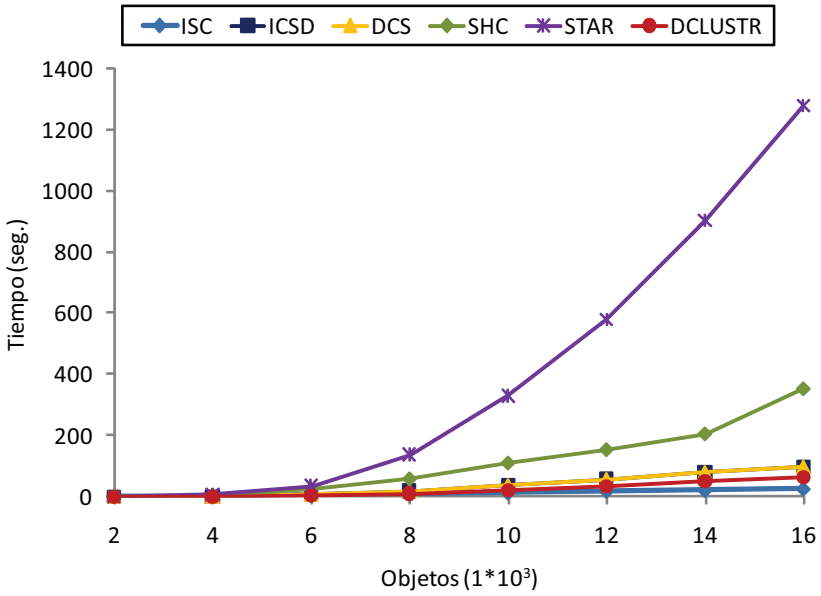
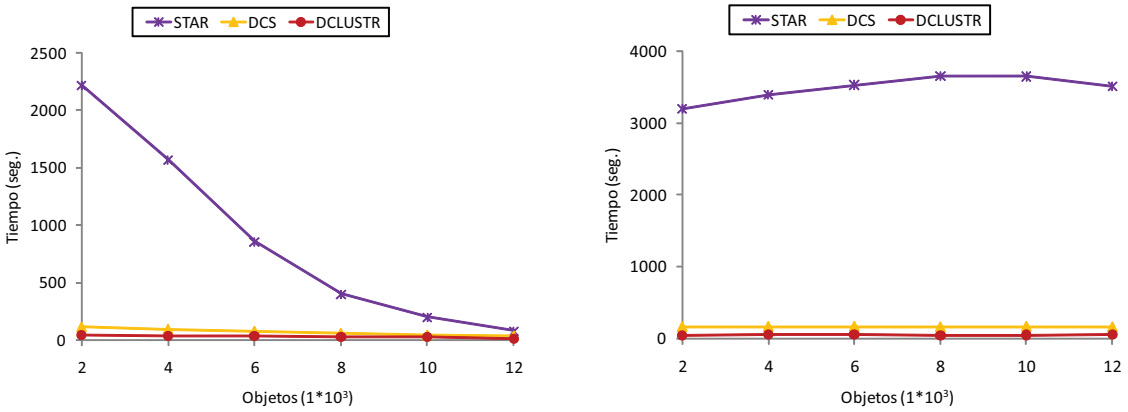


Fig. 2. Tiempo empleado por cada algoritmo para procesar múltiples adiciones sobre TDT.

Como se puede observar en la Figura 5, DClustR tiene un mejor comportamiento que Star, SHC, ICSD y DCS, en el procesamiento de múltiples adiciones sobre la colección TDT. Aunque el comportamiento del algoritmo ISC es ligeramente mejor que el del algoritmo propuesto, es importante mencionar que DClustR produce agrupamientos con mejor calidad que los formados por el algoritmo ISC (ver subsección 3.4.5).

En la Figura 3, se muestra el comportamiento de los algoritmos Star y DOCDC, en el procesamiento de múltiples eliminaciones (Figura 3(a)) y modificaciones (Figura 3(b)).



(a) Procesamiento de múltiples eliminaciones (b) Procesamiento de múltiples modificaciones

Fig. 3. Tiempo empleado por Star, DCS y DClustR para procesar múltiples cambios sobre TDT.

Como se puede observar en la Figura 3, DClustR es más rápido que los algoritmos Star y DCS en el procesamiento de múltiples eliminaciones y modificaciones sobre la colección TDT.

Se realizaron otras pruebas variando el número de documentos que son adicionados, eliminados y modificados en los experimentos, observándose el mismo comportamiento.

3.5. Recapitulación

En esta sección se introdujo un nuevo algoritmo de agrupamiento dinámico, denominado DClustR, que es capaz de formar grupos con traslape. DClustR representa la colección de objetos como un grafo pesado de β-semejanza $\widetilde{G}_\beta$, donde β es un parámetro predefinido. DClustR introduce una nueva estrategia para construir agrupamientos con traslape y una nueva estrategia para actualizar eficientemente dichos agrupamientos, cuando la colección cambia producto de múltiples adiciones/eliminaciones de objetos. Estos aspectos le permiten a DClustR reducir las limitaciones presentes en los algoritmos del estado-del-arte.

El algoritmo propuesto se comparó con los algoritmos del estado-del-arte, que permiten formar grupos con traslape y actualizar dichos grupos cuando la colección cambia. La evaluación se llevó a cabo sobre varias colecciones estándares y se enfocó en comparar los algoritmos de acuerdo: a la calidad de los grupos que forman, el número de grupos, el traslape de los grupos y el tiempo empleado para actualizar el agrupamiento luego de múltiples adiciones y/o eliminaciones. A partir de estos experimentos se puede concluir que DClustR construye agrupamientos traslapados con una mejor calidad, de acuerdo a la medida FBcubed, que los algoritmos del estado-del-arte. Adicionalmente, DClustR forma agrupamientos con menos grupos y menos traslape que los algoritmos utilizados en los experimentos.

Los experimentos también mostraron que la estrategia propuesta por DClustR, para el procesamiento de múltiples adiciones, eliminaciones así como modificaciones de objetos, es claramente más rápida que la usada por los algoritmos dinámicos Star y DCS. Adicionalmente, en los experimentos se mostró que DClustR supera, en el procesamiento de múltiples adiciones, a los algoritmos incrementales SHC y ICSD. Por último, aunque el algoritmo ISC tiene un comportamiento ligeramente mejor que el algoritmo propuesto en el procesamiento de múltiples adiciones, es válido recordar que DClustR supera ampliamente al algoritmo ISC respecto a la calidad de los grupos, el número de grupos construido, así como respecto

al traslape de dichos grupos. Por lo tanto, se puede afirmar que el algoritmo propuesto tiene una mejor relación entre calidad y eficiencia que el algoritmo ISC.

A partir de lo anterior, se puede concluir que DClustR es una mejor opción para enfrentar el problema del agrupamiento traslapado y no jerárquico de objetos, en un contexto dinámico, que los algoritmos no jerárquicos que permiten formar grupos con traslape del estado-del-arte.

4. Algoritmo de agrupamiento jerárquico dinámico con traslape

En esta sección, se introduce el algoritmo DHClustR (del inglés Dynamic Hierarchical Overlapping Clustering based on Relevance). DHClustR es un algoritmo jerárquico y dinámico, que permite obtener una jerarquía de grupos con traslape y actualizarla eficientemente cuadro la colección cambia. La presentación del algoritmo propuesto está dividida en cinco partes. En la subsección 4.1, se describe la estrategia que sigue DHClustR para formar una jerarquía de grupos con traslape. Posteriormente, en la subsección 4.2 se presenta la estrategia que sigue DHClustR para actualizar la jerarquía cuando ocurren múltiples cambios en la colección. En la subsección 4.3, se analiza la complejidad computacional del algoritmo DHClustR. En la subsección 4.4, se describen un conjunto de resultados experimentales en los que se evalúa al algoritmo DHClustR en relación al algoritmo DHS. Finalmente, en la subsección 4.5 se hace una recapitulación de lo presentado en la sección.

4.1. Formando una jerarquía de grupos con traslape

Para organizar los objetos de una colección O en una jerarquía de grupos que pueden tener traslape, DHClustR emplea una estrategia aglomerativa, mediante la cual la jerarquía se construye de abajo hacia arriba; es decir, del nivel más específico al más general. En esta estrategia, los grupos del nivel base de la jerarquía se obtienen producto de la aplicación del algoritmo DClustR sobre los objetos de O. A partir de este punto, el resto de los niveles es construido asumiendo que los objetos de cada nivel son los grupos construidos en el nivel anterior aplicando el algoritmo DClustR. El proceso de construcción de la jerarquía se detiene cuando el grafo pesado de β-semejanza $\widetilde{G}_\beta$, que representa a los objetos del nivel, no tiene aristas. En la Figura 4 se muestra gráficamente el proceso anteriormente descrito.

Los objetos del primer nivel pueden representarse utilizando cualquier modelo. Adicionalmente, para calcular la semejanza entre dichos objetos se puede utilizar cualquier medida de semejanza, siempre que esta sea simétrica (ver sección 2). Por otra parte, dado que los grupos determinados en un nivel son los objetos del nivel siguiente, a continuación se propone un modelo para representar a dichos grupos y una función para calcular la semejanza entre ellos.

4.1.1. Representación de los grupos en la jerarquía

Para la representación de los grupos en la jerarquía el algoritmo DHS utiliza el *vector composición* de cada grupo. El vector composición de un grupo es el vector que se obtiene producto de sumar los vectores que representan a los objetos del grupo. Como se puede apreciar, para utilizar esta representación DHS restringe que los objetos de la colección estén representados como vectores; siendo esta una de sus limitaciones (ver subsección 1.1). Otra limitación de esta representación es que, dado que entre los grupos del un nivel de la jerarquía puede existir traslape, el vector composición de un grupo no puede construirse a partir de los subgrupos que lo componen. Por lo tanto, el vector composición tiene que calcularse usando

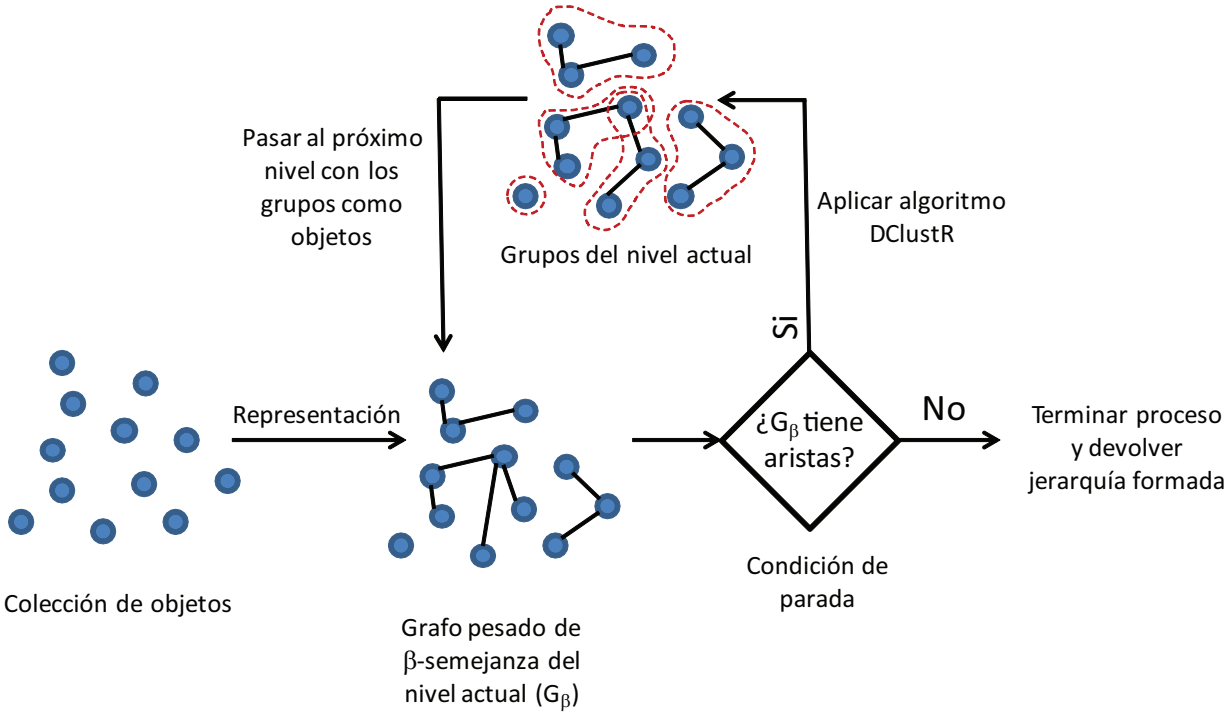


Fig. 4. Proceso de construcción de la jerarquía empleado por el algoritmo DHClustR.

los vectores de los objetos contenidos en el grupo; este proceder es costoso a medida que se avanza la construcción de la jerarquía y los grupos crecen.

La idea al representar un grupo C por el vector de composición, es tener en cuenta en la descripción de C las características de todos los objetos de C. De esta forma, se garantiza que todos los objetos de C participan en el cálculo de semejanza de C con cualquier otro grupo. La idea del modelo propuesto en esta subsección es usar solamente algunos objetos que tienen un significado especial para C. Lo anterior permitiría reducir el número de elementos de C que participen en el cálculo de la semejanza entre C y el resto de los grupos.

Sea $\widetilde{G}_\beta$ el grafo pesado de β-semejanza que representa a los objetos en un nivel de la jerarquía. Sea C un grupo construido por DClustR al procesar $\widetilde{G}_\beta$ y v_1 el vértice marcado como *semilla* que determina a C. Sea $P = \{p_1, p_2, \ldots, p_m\}$ un conjunto de vértices, donde cada vértice $p_i, i = 1 \ldots m$, cumple las siguientes condiciones:

- $p_i \in C$
- El ws-grafo determinado por p_i ($G^\star_{p_i}$) fue seleccionado por DClustR en la fase de inicialización, pero eliminado en la de perfeccionamiento por no ser $G^\star_{p_i}$ útil para el cubrimiento de $\widetilde{G}_\beta$
- Los vértices de $p_i.Non\_shared$ fueron adicionados a la lista to $v_1.Linked$

El *núcleo* del grupo C, denotado por $Cr(C)$, es la tupla formada por el vértice v_1 y los vértices del conjunto P; es decir, $Cr(C) = (v_1, p_1, p_2, \ldots, p_m)$. De esta forma, se representa al grupo C usando dos tipos de vértices que son importantes para C: i) el vértice que define al grupo (v_1 en este caso) y ii) aquellos vértices que son semejantes a v_1 y que adicionaron vértices al grupo C, a través de $v_1.Linked$. Note que el resto de los objetos que pertenecen a C eran adyacentes al propio vértice v_1. Es importante notar que el núcleo del grupo C se puede construir mientras se está construyendo el propio grupo C, sin

afectar la complejidad del algoritmo DClustR. Adicionalmente, a diferencia del vector composición, la definición de núcleo de un grupo no depende del espacio de representación de los objetos.

4.1.2. Cálculo de la semejanza entre dos grupos

Como se mencionó en la subsección 1.1, el algoritmo DHS utiliza la medida *group-average* de conjunto con la medida del coseno, con el objetivo de acelerar el cálculo de semejanza entre dos grupos. Sin embargo, este uso introduce una restricción en el uso de este algoritmo y, por lo tanto, reduce el alcance de su aplicación. Es importante comprender que, en asco de que DHS no utiliza esta restricción, entonces para el cálculo de la semejanza entre dos grupos tiene que calcular la semejanza entre $|C_1| \cdot |C_2|$ pares de objetos. Es decir, la no utilización de esta restricción implica un incremento en el tiempo de procesamiento de DHS. Para resolver esta limitación de DHS, se propone calcular la semejanza entre dos grupos a través de sus núcleos.

Sea $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, S \rangle$ el grafo pesado de β-semejanza que representa a los objetos del nivel i. Sea C_1 y C_2 dos grupos construidos en el nivel i y $Cr(C_1) = (v, p_1, p_2, \ldots, p_m)$ y $Cr(C_2) = (u, q_1, q_2, \ldots, q_t)$ los núcleos de C_1 y C_2, respectivamente. Sea $(w, o) \in \widetilde{E}_\beta$ la arista que conecta un vértice $w \in Cr(C_1)$ con un vértice $o \in Cr(C_2)$. Esta arista se clasifica de acuerdo a las siguientes condiciones:

i) **Tipo A**: cuando la arista conecta a los vértices que definen a ambos grupos; es decir, a los vértices v y u.
ii) **Tipo B**: cuando la arista conecta al vértice v con un vértice del conjunto $\{q_1, q_2, \ldots, q_t\}$ o cuando conecte al vértice u con un vértice del conjunto $\{p_1, p_2, \ldots, p_m\}$.
iii) **Tipo C**: cuando la arista conecta un vértice del conjunto $\{q_1, q_2, \ldots, q_t\}$ con otro del conjunto $\{p_1, p_2, \ldots, p_m\}$.

Es importante notar que entre los núcleos de C_1 y C_2 solo puede existir una sola arista de tipo A. Dado que esta arista conecta los dos vértices que definen ambos grupos, sería lógico ver a esta arista como la más importante de todas las aristas que pueden existir entre los núcleos C_1 y C_2. Por otra parte, las aristas de tipo B conectan al vértice que define a un grupo con algunos de los vértices que son semejantes al vértice que define a otro grupo. Luego, las aristas de tipo B podrían verse como las segundas más importantes entre $Cr(C_1)$ y $Cr(C_2)$; el número máximo de aristas de tipo B que puede existir entre $Cr(C_1)$ y $Cr(C_2)$ es $(|Cr(C_1)| - 1) + (|Cr(C_2)| - 1)$. Finalmente, una arista de tipo C conecta dos vértices que son semejantes a los vértices definen a cada grupo. Por lo tanto, este tipo de aristas será visto como las menos importantes entre todas las aristas que existen entre los núcleos de dichos grupos; el número máximo de aristas de tipo C que puede existir entre $Cr(C_1)$ y $Cr(C_2)$ es $(|Cr(C_1)| - 1) * (|Cr(C_2)| - 1)$.

Las aristas existentes entre los núcleos de C_1 y C_2 representan enlaces o puntos de coincidencia entre ambos grupos. La idea que se propone para calcular la semejanza entre C_1 y C_2 es utilizar el número y peso de los tres tipos de aristas que pueden existir entre $Cr(C_1)$ y $Cr(C_2)$. De esta forma, la estrategia sería calcular cuán semejantes son C_1 y C_2 de acuerdo a cada tipo de aristas y, con base en estos valores, calcular un valor de semejanza global.

Sea N_B y N_C el conjunto de aristas de tipo B y C, que existen entre los núcleos de C_1 y C_2. Sea β el umbral de semejanza utilizado para construir el grafo $\widetilde{G}_\beta$ del primer nivel de la jerarquía. La semejanza entre dos grupos C_1 y C_2, de cualquier nivel de la jerarquía, se denota por $S_T(C_1, C_2)$ y se calcula de la siguiente forma:

$$S_T(C_1, C_2) = \frac{S_A(C_1, C_2) + S_B(C_1, C_2) + S_C(C_1, C_2)}{3}, \tag{16}$$

donde:

- $S_A(C_1,C_2)$ es el valor de semejanza que existe entre C_1 y C_2, considerando solamente las aristas de tipo A que existen entre $Cr(C_1)$ y $Cr(C_2)$. Así, $S_A(C_1,C_2) = S(v,u)$ ssi existe la arista $(v,u) \in \widetilde{E}_\beta$; en otro caso, $S_A(C_1,C_2) = 0$
- $S_B(C_1,C_2)$ es el valor de semejanza que existe entre C_1 and C_2, considerando solamente las aristas del conjunto N_B. Luego, este valor se calcula de la siguiente forma:

$$S_B(C_1,C_2) = \begin{cases} \frac{\sum_{e\in N_B} S(e)}{|N_B|} \text{ ssi } (|Cr(C_1)|>1 \text{ ó } |Cr(C_2)|>1) \\ \qquad\qquad y \\ \qquad\qquad \frac{|N_B|}{(|Cr(C_1)|-1)+(|Cr(C_2)|-1)} \geq \beta \qquad , \\ 0 \qquad\qquad \text{en otro caso} \end{cases} \tag{17}$$

- $S_C(C_1,C_2)$ es el valor de semejanza que existe entre C_1 and C_2, considerando solamente las aristas del conjunto N_C. Luego, este valor se calcula como sigue:

$$S_C(C_1,C_2) = \begin{cases} \frac{\sum_{e\in N_C} S(e)}{|N_C|} \text{ ssi } |Cr(C_1)|>1 \text{ y } |Cr(C_2)|>1 \\ \qquad\qquad y \\ \qquad\qquad \frac{|N_C|}{(|Cr(C_1)|-1)*(|Cr(C_2)|-1)} \geq \beta \qquad , \\ 0 \qquad\qquad \text{en otro caso} \end{cases} \tag{18}$$

Como puede notarse en las ecuaciones (17) y (18), el valor de β se utiliza para determinar cuándo el número de aristas de tipo B y C, que existen entre $Cr(C_1)$ y $Cr(C_2)$, es lo suficientemente significativo para concluir que estos grupos son semejantes de acuerdo a estos tipos de aristas. Así, si el cociente entre el número de aristas de tipo B y el número máximo de aristas de este tipo que podrían existir entre $Cr(C_1)$ y $Cr(C_2)$ es mayor que β, entonces la semejanza entre C_1 y C_2 de acuerdo a este tipo de aristas (*i.e.*, $S_B(C_1,C_2)$) es el promedio de peso existente entre dichas aristas; en otro caso, el valor de semejanza es cero. Este mismo análisis se aplica a las aristas de tipo C.

Con base en el análisis anterior, se plantea que dos grupos C_1 y C_2 se consideran *semejantes* ssi $S_T(C_1,C_2) > 0$; en otro caso, se dirá que los grupos no son semejantes. Es importante notar que el criterio propuesto para calcular la semejanza entre dos grupos, no depende en el espacio de representación de los objetos de la colección ni en la medida de semejanza utilizada para calcular la semejanza entre dichos objetos (función S). Además, dado que la fórmula propuesta utiliza los valores de semejanza calculados en el nivel previo, el cálculo de la semejanza entre dos grupos no es computacionalmente costoso.

4.2. Actualización de la jerarquía después de los cambios

Sea $J_G = \{\widetilde{G}_\beta^{(1)}, \widetilde{G}_\beta^{(2)}, \ldots, \widetilde{G}_\beta^{(k)}\}$ el conjunto donde $\widetilde{G}_\beta^{(i)}, i = 1 \ldots k$, es el grafo pesado de β-semejanza, que representa a los objetos del nivel i de la jerarquía formada por DHClustR, siguiendo la estrategia presentada en la subsección 4.1. Sea $J_C = \{SC_1, SC_2, \ldots, SC_k\}$ el conjunto donde $SC_i, i = 1 \ldots k$, es el agrupamiento formado por DHCLustR en cada nivel de la jerarquía.

Al adicionar, eliminar o modificar objetos de la colección, el grafo $\widetilde{G}_\beta^{(1)} = \langle V^{(1)}, \widetilde{E}_\beta^{(1)}, S\rangle$ que representaba a los objetos del primer nivel tiene que ser actualizado. La actualización de $\widetilde{G}_\beta^{(1)}$ puede provocar que se adicionen y/o eliminen aristas y por lo tanto, que se necesite actualizar el agrupamiento del primer nivel (SC_1). De la actualización del agrupamiento del primer nivel se encarga el algoritmo DClustR.

Como se explicó en la sección 4.1, los grupos obtenidos en un nivel son los objetos agrupados en el nivel siguiente. Luego, cuando el agrupamiento del nivel base de la jerarquía es actualizado, se eliminan y/o adicionan grupos. Los grupos adicionados al agrupamiento representan objetos que deben ser adicionados a la colección de objetos del segundo nivel. Por otra parte, los grupos eliminados constituyen objetos que deben ser eliminados del segundo nivel. Finalmente, los grupos modificados representan objetos que deben ser eliminados del segundo nivel y vueltos a añadir, con sus características actuales.

Los grupos adicionados (grupos *nuevos* en el agrupamiento) son aquellos determinados por: 1) vértices que fueron adicionados a la colección o 2) vértices que no estaban marcados como *semilla* antes de actualizar el agrupamiento, pero que luego de dicha actualización si lo estaban. Los grupos eliminados son aquellos que estaban determinados por: 1) vértices eliminados de la colección o 2) vértices que estaban marcados como *semilla*, pero que luego de la actualizar el agrupamiento no lo están. Estos dos tipos de grupos son fáciles de detectar en el mismo proceso de creación de los grupos que realiza DClustR, sin afectar la complejidad de DClustR. Por otra parte, para saber cómo detectar eficientemente los grupos modificados es necesario un análisis más profundo.

Sea v un vértice que está marcado como *semilla*, luego de la actualización del agrupamiento del nivel. Sea C_1 el grupo determinado por v y $Cr(C_1) = (v, p_1, p_2, \ldots, p_m)$ el núcleo de C_1. Para considerar el grupo C_1 como *modificado* una vez termina el proceso de actualización del agrupamiento, C_1 deberá satisface las siguientes condiciones:

i) C_1 no es un grupo nuevo; es decir, el vértice v estaba marcado como semilla antes del proceso de actualización del agrupamiento del nivel.
ii) Existe al menos otro grupo C_2, que satisface también la condición anterior, tal que la semejanza entre C_2 y C_1 puede haber cambiado luego de la actualización del agrupamiento.

La condición i) es fácil de verificar. Como la semejanza entre C_1 y $C2$ se calcula a través de la función $S_T(C_1,C_2)$ y esta función utiliza para su cálculo los núcleos de dichos grupos, para verificar la condición ii) se debe analizar los cambios que puedan haber ocurrido en $Cr(C_1)$, luego de la actualización del agrupamiento. De acuerdo a la ecuación (16), el valor de $S_T(C_1,C_2)$ puede cambiar solo si el valor de $S_B(C_1,C_2)$ o $S_C(C_1,C_2)$ cambia; el valor de $S_A(C_1,C_2)$ no cambia si tanto C_1 como C_2 no son grupos nuevos. Observe que, como C_1 y C_2 no son grupos nuevos, si existiera una arista entre los vértices que determinan a ambos grupos, esta no cambia luego de la actualización. De forma análoga, si no existiera una arista entre los vértices que determinan a ambos grupos, esta tampoco aparecería luego de la actualización.

Analizando la ecuación (17), se puede concluir que el valor de $S_B(C_1,C_2)$ puede cambiar si N_B o $(|Cr(C_1)|-1)+(|Cr(C_2)|-1)$ cambian. N_B cambia si se adicionan/eliminan aristas de tipo B entre C_1 y C_2; esto es posible solo si algunos vértices fueron adicionados/eliminados de $Cr(C_1)$ o de $Cr(C_2)$. Análogamente, la expresión $(|Cr(C_1)|-1)+(|Cr(C_2)|-1)$ cambia solamente si algunos vértices se adicionan/eliminan de $Cr(C_1)$ o de $Cr(C_2)$. Analizando la ecuación (18) se llega también a la conclusión anterior. Por lo tanto, los grupos modificados serán aquellos que no sean nuevos y en cuyos núcleos se hayan adicionado/eliminado algún vértice. La detección de este tipo de grupos puede también hacerse en el mismo proceso de creación de los grupos que realiza DClustR, sin afectar la complejidad de DClustR.

Sea R y A, el conjunto de grupos eliminados y adicionados luego de actualizar el agrupamiento. Como se comentó en la subsección 1.1, una modificación se trata como una eliminación seguida de una adición; por lo tanto, los grupos modificados pertenecen tanto a R como a A. Una vez que el agrupamiento en el primer nivel fue actualizado, el grafo que representa a la colección del segundo nivel ($\widetilde{G}_{\beta}^{(2)}$) es actualizado, utilizando los grupos en R y en A como objetos a eliminar y adicionar, respectivamente; para esta acción se puede utilizar el procedimiento *UpdCovCompt* (ver Algoritmo 2). Cuando se actualiza el grafo del segundo nivel, el agrupamiento se actualiza utilizando el algoritmo DClustR y todo el proceso anterior

vuelve a repetirse. Así, cada vez que se actualiza el agrupamiento de un nivel *i* de la jerarquía, el nivel siguiente debe ser revisado, pues puede que haya cambiado y que por lo tanto, necesite ser actualizado.

El proceso de actualización de la jerarquía continua hasta que se alcanza la condición de parada del algoritmo; *i.e.*, el grafo de β-semejanza pesado que representa a los objetos del nivel no tiene aristas. Si se alcanza la condición de parada antes de llegar al tope de la jerarquía, los niveles siguientes son eliminados; el tope de la jerarquía es el último nivel de la misma. Por otra parte, si se alcanza el tope antes de la condición de parada, la jerarquía sigue construyéndose según la idea general propuesta en la subsección 4.1.

El pseudocódigo del algoritmo DHClustR se muestra en el Algoritmo 4. Para actualizar el grafo de cada nivel se utiliza el procedimiento *UpdCovCompt* (ver Algoritmo 2). Para actualizar el agrupamiento de cada nivel se utiliza el algoritmo DClustR, sin el primer paso (ver Algoritmo 1). Este primer paso es el llamado al procedimiento *UpdCovCompt*, que ya se hace como parte del código de DHClustR. Es importante que este cambio en el código de DClustR no afecta para nada su complejidad computacional.

Algoritmo 4: Algoritmo DHClustR

Input: O - Colección de objetos, J_G - Jerarquía de grafos, J_C - Jerarquía de agrupamientos, β - Umbral de semejanza, A - Conjunto de objetos a adicionar, R - Conjunto de objetos a eliminar
Output: O, J_G, J_C

```
1  i := 1;
2  G̃β := JG[i];
3  M := ∅;
4  UpdCovCompt(O,G̃β,β,A,R,M);
5  repeat
6      if i ≠ 1 y G̃β es el nivel tope then
7          Z := {v | v ∈ V ∧ |v.Adj| = 0};
8          M := M ∪ Z;
9      end
10     SC := ∅;
11     A := ∅;
12     R := ∅;
13     DClustR(G̃β,SC, M);
14     "Construir A y R utilizando SC y JC[i]" ;        // Objetos a ser adicionados/eliminados en el nivel
       siguiente
15     JC[i] := SC;
16     JG[i] := G̃β;
17     i++;
18     G̃β := JG[i];
19     O' := V;
20     M := ∅;
21     UpdCovCompt(O',G̃β,β,A,R,M);
22 until |Ẽβ| = 0 ;
23 if El nivel tope anterior no fue alcanzado then "Eliminar los niveles restantes";
```

Como se mencionó anteriormente, los objetos del primer nivel pueden representarse utilizando cualquier modelo y en el cálculo de la semejanza entre dichos objetos se puede utilizar cualquier medida de semejanza, siempre que esta sea simétrica (ver sección 2). Por otra parte, para representar a los objetos del resto de los niveles y para calcular la semejanza entre ellos, se utilizan el modelo propuesto en la subsección 4.1.1 y la función propuesta en la subsección 4.1.2, respectivamente.

Como se puede observar en el código de Algoritmo 4, DHClustR asume que existen dos estructuras que contienen toda la información de la jerarquía. Una de estructuras es J_G, que contiene a los grafos pesados de β-semejanza que representan a la colección de objetos de cada nivel de la jerarquía. La otra estructura es J_C, que contiene a los agrupamientos formados en cada nivel de la jerarquía. Es importante

entender que, si no existiese una colección previa (es decir, si $O = \emptyset$) y esta es la primera vez que se agrupará la colección, entonces $J_G = \emptyset$ y $J_C = \emptyset$; de esta forma, DHClustR puede procesar también una colección empezando desde cero.

Es importante mencionar que, aunque la estrategia propuesta por DHClustR es similar a la del algoritmo DHS, existen varias diferencias entre ellas que se deben resaltar. Primero que todo, DHClustR resuelve las limitaciones presentes en DHS: i) DHClustR no impone restricciones al espacio de representación de los objetos de la colección, ni a la medida de semejanza entre dichos objetos, y ii) DHClustR no deja elementos sin agrupar en ningún nivel de la jerarquía.

Otra diferencia es que la estrategia de DHClustR solo necesita construir un grafo por cada nivel, mientras que la estrategia que emplea DHS necesita construir dos grafos: un grafo de β-semejanza para representar a los objetos del nivel y un grafo de máxima β-semejanza para construir los grupos. Note que la construcción de este segundo grafo implica tiempo adicional de procesamiento por parte de DHS.

Tercero, a diferencia de DHS que utiliza una variante del algoritmo Star [28] para formar el agrupamiento de los niveles, DHClustR utiliza el algoritmo DClustR para formar el agrupamiento de cada nivel. DClustR mostró que forma agrupamientos de mejor calidad que los que forma el algoritmo Star (ver subsección 3.4.3). Adicionalmente, como se mostrará en los experimentos, la estrategia de agrupamiento jerárquico de DHClustR le permite formar jerarquías de grupos, con una mejor calidad que las construidas por DHS. Por último, como se verá en la próxima subsección, DHClustR tiene una menor complejidad computacional que DHS, lo que le permite procesar una colección en menos tiempo que lo que le tomaría a DHS procesar la misma colección.

Finalmente, es necesario mencionar que el algoritmo DHClustR depende del orden de análisis de los objetos; es decir, DHClustR podría construir diferentes jerarquías de grupos, a partir de una misma colección y con mismos parámetros, dependiendo del orden en que se analicen los objetos. No obstante, como se menciona en la subsección 4.4, la desviación estándar entre las calidades de estas diferentes jerarquías es pequeña.

4.3. Análisis de complejidad computacional

Para determinar la complejidad computacional del algoritmo DHClustR, se analizarán cada uno de sus pasos. Sea $n_A = |A|$ y $n_R = |R|$ el número de objetos adicionados y eliminados, respectivamente. Sea n_0 y n el número de vértices de $\widetilde{G}_\beta$ antes y después de procesar los cambios en el primer nivel de la jerarquía, respectivamente.

El número de operaciones realizadas en los primeros cuatro pasos del algoritmo DHClustR están determinadas por el número de operaciones realizadas en el paso 4. Según el análisis de la subsección 3.3 este número es $T_{1-4}(n) = n^2$; por tanto, $T_{1-4}(n)$ es $O(n^2)$. El número de operaciones realizadas en los pasos 5-23 está determinado por el número de operaciones realizadas en los pasos 5-23, el cual depende de dos factores: el número de operaciones realizadas en los pasos 6-21 para cada nivel de la jerarquía y el número de niveles que se construyen.

Sea $n_i \leq n$ el número de objetos en un nivel de la jerarquía. El número de operaciones realizadas en los pasos 6-12 está determinada por las operaciones realizadas en los pasos 6-9, siendo este número $T_{6-12}(n_i) = n_i$; luego, $T_{6-12}(n_i)$ es $O(n_i)$. A partir del análisis presentado en la subsección 3.3, se puede concluir que el número de operaciones realizadas en el paso 13 es $T_{13}(n_i) = n_i^2$; luego, $T_{13}(n_i)$ es $O(n_i^2)$. Como se comentó anteriormente, la detección de los grupos adicionados, eliminados y modificados puede hacerse durante la ejecución del algoritmo DClustR (paso 13), sin afectar la complejidad computacional del mismo. Luego, el número de operaciones realizadas en el paso 14 es $T_{14}(n_i) = |SC| + |J_C[i]|$. En el caso peor, el número de grupos en un nivel es n_i. Luego, $T_{14}(n_i) = 2 \cdot n_i$ y por consiguiente, $T_{14}(n_i)$ es

$O(n_i)$. El número de operaciones de los pasos 15-21 está determinado por el número de operaciones del paso 21 que, como se explicó anteriormente, es cuadrático; por lo tanto, $T_{15-21}(n_i)$ es $O(n_i^2)$. A partir del análisis anterior, se tiene que el número de operaciones realizadas en los pasos 6-21 es $T_{6-21}(n_i) = T_{6-12}(n_i) + T_{13}(n_i) + T_{14}(n_i) + T_{15-21}(n_i) = n_i + n_i^2 + n_i + n_i^2$; por lo tanto, $T_{6-21}(n_i) = 2 \cdot n_i^2 + 2 \cdot n_i$.

Como se mencionó anteriormente, el número máximo de grupos que se pueden crear en un nivel es n_i. No obstante, esto significa que todos los objetos forman grupos aislados y por lo tanto, se detiene el proceso de formación de la jerarquía. La mayor cantidad de niveles de la jerarquía se forma cuando en cada nivel se obtiene la menor cantidad de grupos unitarios; es decir, formados por ws-grafos degenerados. Este caso ocurre cuando el número de vértices marcados como semilla en cada nivel es $\frac{n_i}{2}$. Note que, como todos los vértices del grafo están cubiertos, obtener un número mayor de grupos significa que dichos grupos son unitarios. En otro caso, si estos grupos no fueran unitarios, entonces los vértices marcados como semilla que los determinan formarían ws-grafos no útiles. Esto implica que dichos grupos debieron ser eliminados en la etapa de perfeccionamiento de DClustR. Sea t el número de niveles construidos y $n_1, n_2, \ldots, n_t$ el número de objetos de cada nivel. Como conclusión del análisis anterior se tiene que, en el peor de los casos, $t = \log_2 n$ y que $n_i = \frac{n_{i-1}}{2}, i = 2 \ldots t$.

Hasta este punto, se tiene que el número de operaciones realizadas en los pasos 5-22 es:

$$T_{5-22}(n) = \sum_{i=1}^{\log_2 n} T_{6-21}(n_i) = \sum_{i=1}^{\log_2 n} (2 \cdot n_i^2 + 2 \cdot n_i) = 2 \cdot \sum_{i=1}^{\log_2 n} n_i^2 + 2 \cdot \sum_{i=1}^{\log_2 n} n_i,$$

con base en el análisis del párrafo anterior:

$$T_{5-22}(n) = 2 \cdot (n_1^2 + \frac{n_1^2}{2} + \frac{n_1^2}{4} + \ldots + \frac{n_1^2}{2^{\log_2 n}}) + 2 \cdot (n_1 + \frac{n_1}{2} + \frac{n_1}{4} + \ldots + \frac{n_1}{2^{\log_2 n}}),$$

como $n_1 = n$, se tiene que

$$T_{5-22}(n) = 2 \cdot (n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \ldots + \frac{n^2}{2^{\log_2 n}}) + 2 \cdot (n + \frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^{\log_2 n}}). \tag{19}$$

Como se puede observar en (19), cada término de $T_{5-22}(n)$ es una progresión geométrica, por lo tanto:

$$T_{5-22}(n) = 2 \cdot \left(\frac{\frac{n^2}{2^{\log_2 n}} \cdot \frac{1}{2} - n^2}{\frac{1}{2} - 1} \right) + 2 \cdot \left(\frac{\frac{n}{2^{\log_2 n}} \cdot \frac{1}{2} - n}{\frac{1}{2} - 1} \right),$$

$$T_{5-22}(n) = 2 \cdot \left(\frac{\frac{n^2}{n} \cdot \frac{1}{2} - n^2}{-\frac{1}{2}} \right) + 2 \cdot \left(\frac{\frac{n}{n} \cdot \frac{1}{2} - n}{-\frac{1}{2}} \right) = 2 \cdot \left(2 \cdot n^2 - n\right) + 2 \cdot (2 \cdot n - 1),$$

$$T_{5-22}(n) = 4 \cdot n^2 + 2 \cdot n - 2. \tag{20}$$

A partir de (20) se puede concluir que $T_{5-2}(n)$ es $O(n^2)$. Finalmente, el número de operaciones realizadas en el Algoritmo 4 es $T_t(n) = T_{1-4}(n) + T_{5-22}(n)$. Por la regla de la suma, se tiene que $T_t(n)$ es $O(T_{1-4}(n), T_{5-22}(n))$ y por lo tanto, $T_t(n)$ es $O(n^2)$. Luego, la complejidad computacional del algoritmo DHClustR es $O(n^2)$. Como se puede observar, la complejidad computacional del algoritmo propuesto es menor que la del algoritmo DHS, la cual es $O(n^3)$. Esto le permitirá al algoritmo DHClustR ser capaz de procesar una colección en menos tiempo de lo que le tomaría al algoritmo DHS procesar la misma colección.

4.4. Resultados experimentales

En esta sección, se presentan los resultados de un conjunto de experimentos en los que se evalúa al algoritmo DHClustR de acuerdo a varios criterios. Los experimentos fueron realizados sobre las colecciones descritas en la subsección 3.4.1. En estos experimentos se compararon los resultados obtenidos por DHClustR contra los resultados obtenidos por el algoritmo DHS [34]. Com se explicó en la subsección 1.1, DHS es el único algoritmo jerárquico que permite formar jerarquías traslapadas y que es capaz de actualizar dicha jerarquía luego de cambios en la colección.

El primer experimento se concentró en evaluar los algoritmos de acuerdo a la calidad de la jerarquía que construyen. El segundo experimento se enfocó en evaluar la jerarquía formada por ambos algoritmos, de acuerdo al número de niveles y el número de grupos contenidos en la misma. Por último, el tercer experimento se enfocó en evaluar la eficiencia de los algoritmos en el procesamiento de varias colecciones de prueba. Los aspectos evaluados en cada experimentos han sido los utilizados en varios trabajos para evaluar algoritmos jerárquicos [50,51,34].

Los algoritmos usados en los experimentos fueron implementados en C++ y compilados utilizando el compilador G++. Los experimentos se realizaron en una computadora con un procesador Intel Core 2 Duo a 1.86 GHz, 2 GB de memoria RAM y con sistema operativo RedHat Enterprise Linux 5.3.

4.4.1. Evaluación de algoritmos jerárquicos

Una estrategia para determinar la calidad de la jerarquía formada por un algoritmo consiste en evaluar, utilizando el *ground-truth* de la colección y alguna de las medidas externas mencionadas en la subsección 3.4.2, cada uno de los niveles de dicha jerarquía. El nivel que tenga el máximo valor de la medida de calidad utilizada, se considera como el *mejor* nivel de la jerarquía; el valor de calidad de este nivel se considera como el valor de calidad de la jerarquía. Para comparar las jerarquías formadas por dos algoritmos jerárquicos, se determina el mejor nivel de cada jerarquía y posteriormente, se comparan los valores de calidad de dichos niveles. El mejor algoritmo será aquel que construya una jerarquía cuyo mejor nivel tenga el mayor valor de calidad.

Esta forma de evaluar los algoritmos jerárquicos, en lo siguiente referida como alternativa 1, realmente lo que hace es transformar la evaluación de una jerarquía en la evaluación de varios agrupamientos obtenidos por un algoritmo de agrupamiento. Luego, las limitaciones de esta forma de evaluar los algoritmos jerárquicos, son las mismas que tenga la medida externa utilizada para determinar el mejor nivel. En la subsección 3.4.2 se describieron las limitaciones de las medidas externas más utilizadas. Con base en lo anterior y conociendo que se quiere evaluar jerarquías que tienen agrupamiento con traslape como niveles, una medida externa apropiada para utilizar en esta estrategia sería la FBcubed.

Otra estrategia que se ha utilizado, para evaluar la calidad de una jerarquía, es *aplanar la jerarquía* y evaluar el conjunto de grupos resultante con alguna medida externa. En este contexto, *aplanar la jerarquía* es considerar como salida del algoritmo al conjunto formado por todos los grupos de cada nivel de la jerarquía. Una vez que se aplana la jerarquía, el conjunto resultante es evaluado utilizando una medida de evaluación externa. La medida de evaluación que se utilice debe permitir evaluar grupos traslapados pues, el conjunto de grupos obtenidos al aplanar una jerarquía es traslapado. El mejor algoritmo será aquel que construya una jerarquía que al aplanarse tenga el mayor valor de calidad.

Esta forma de evaluación, en lo siguiente referida como alternativa 2, transforma el problema de evaluar una jerarquía de grupos en el problema de evaluar un conjunto de grupos traslapados. Las limitaciones de esta alternativa de evaluación, están determinadas por las limitaciones intrínsecas de la medida externa utilizada. Adicionalmente, el mismo proceso de aplanado de la jerarquía aumenta las limitaciones de

las medidas pues, con este proceso, aumenta tanto el número de grupos como el traslape del conjunto resultante.

Otra alternativa de evaluación fue propuesta en [52], bajo el marco del proyecto TDT de la Universidad de Massachusetts [53]. Para evaluar un algoritmo jerárquico en esta alternativa, se realiza una búsqueda en profundidad por la jerarquía formada por dicho algoritmo. El objetivo de esta búsqueda es encontrar el grupo de la jerarquía que mejor se coteja con un grupo del *ground-truth*. Para determinar el cotejo de un grupo de la jerarquía con uno del *ground-truth* se utiliza la medida de *costo de detección* o CDET [52]. Como parte de la evaluación, esta alternativa incluye además el costo de navegar desde la raíz de la jerarquía hasta el grupo que mejor se coteja.

Esta forma de evaluación, en lo siguiente referida como alternativa 3, tiene algunas limitaciones. Primero, la medida CDET utilizada es una medida basada en cotejo de conjuntos; por lo tanto, tiene las mismas limitaciones que la F1-measure [42] (ver subsección 3.4.2). Adicionalmente, el uso de esta alternativa de evaluación exige ajustar valores para varios parámetros utilizados en el cálculo del costo de navegación. El ajuste de estos parámetros depende de las características de cada colección y generalmente se hace *ad-hoc*.

En [34], se propuso una alternativa que evalúa un algoritmo jerárquico mediante la comparación de las relaciones entre los nodos de la jerarquía construida por el algoritmo y las que debieran existir, de acuerdo a un *ground-truth* jerárquico de la colección. Esta estrategia necesita que la colección esté etiquetada jerárquicamente; es decir, tener un *ground-truth* jerárquico.

Esta alternativa de evaluación, en lo siguiente referida como alternativa 4, es reciente y por lo tanto no ha sido evaluada y analizada lo suficiente como para conocer en detalle todas sus limitaciones. No obstante, dado que esta estrategia utiliza la medida F1-measure, pudiera tener las mismas limitaciones que la medida F1-measure.

De las alternativas descritas, la que ha sido más utilizada en la literatura es la alternativa 1 [51,54,34]. Por tal motivo, se ha decidido utilizar la alternativa 1 para evaluar las jerarquías formadas por los algoritmos DHClustR y DHS.

4.4.2. Calidad de la jerarquía

En este experimento se comparan los algoritmos DHClustR y DHS, en cuanto a la calidad de las jerarquías que ambos construyen. Para calcular la calidad de estas jerarquías se utilizó la alternativa 1 descrita en la subsección anterior, de conjunto con la medida FBcubed descrita en al subsección 3.4.2. Para una mejor comprensión de cómo se llevó a cabo este experimento, a continuación se describe el procedimiento utilizado sobre la colección AFP.

Primeramente, ambos algoritmos fueron ejecutados sobre la colección AFP, usando $\beta = 0{,}05$. A continuación, se calculó el valor de FBcubed de cada uno de los niveles de las jerarquías formadas por cada algoritmo. Así, el valor de calidad de la jerarquía construida por cada algoritmo para $\beta = 0{,}05$, es el valor de FBcubed del nivel con mayor valor de FBcubed en toda la jerarquía. En el caso del algoritmo DHS, este valor de calidad calculado representa su mejor resultado para $\beta = 0{,}05$. Como DHClustR depende del orden de análisis de los objetos, se repitió el experimento anterior veinte veces, variando los órdenes de los documentos de AFP. En el caso del algoritmo DHClustR, el promedio de calidad de todas las jerarquías formadas para los distintos órdenes representa el mejor valor de calidad para $\beta = 0{,}05$. El experimento anterior fue repetido usando valores de β en el intervalo [0.05,0.35], con un incremento de 0.01; es decir, se utilizó β=0.05,0.06,0.07...,0.35. Para cada uno de estos valores, se determinó el valor de calidad de la jerarquía formada. Finalmente, el mayor valor de calidad alcanzado por cada algoritmo corresponde con la mejor evaluación de ambos algoritmos para la colección AFP.

El procedimiento anteriormente descrito se utilizó también con el resto de las colecciones. Durante estas evaluaciones se observó lo siguiente:

- Para valores de β mayores que 0.35 o menores de 0.05, la calidad de la jerarquía formada por cada algoritmo decreció; por esta razón no se utilizaron valores de β fuera de este intervalo.
- Aún cuando el algoritmo DHClustR depende del orden de análisis de los objetos, la desviación estándar del valor de FBcubed de las jerarquías formadas para los diferentes órdenes de un mismo valor de β, fue menor de 0.01, para cada valor de β usado. Por lo tanto, se puede utilizar el valor promedio de FBcubed de estas diferentes jerarquías, formadas sobre una colección, utilizando un mismo valor de β, como la mejor evaluación de DHClustR sobre esa colección, para dicho valor de β.
- En la mayoría de los casos, el nivel de mejor evaluación fue el nivel tope de la jerarquía.

En la Tabla 6 se muestra la mejor evaluación de DHClustR y DHS, sobre cada colección de prueba.

Tabla 6. Mejor evaluación de ambos algoritmos sobre cada una de las colecciones. El valor más alto por colección aparece en negrita.

	Colecciones					
Algoritmos	AFP	Reu-Te	Reu-Tr	Reuter	TDT	cacm
DHS	**0.80**	0.49	0.44	0.42	0.45	0.29
DHClustR	0.77	**0.53**	**0.46**	**0.45**	**0.49**	**0.35**
	Colecciones					
Algoritmos	TDT-1	TDT-2	TDT-3	TDT-4	TDT-5	cisi
DHS	0.45	0.47	0.48	0.48	0.48	0.29
DHClustR 2	**0.49**	**0.53**	**0.51**	**0.52**	**0.51**	**0.34**

Como puede observarse en la Tabla 6, el algoritmo DHClustR obtiene los valores de calidad más altos, en la mayoría de las colecciones. Para resumir los resultados mostrados en esta tabla, se determinó la significancia estadística de los resultados obtenidos por DHClustR, en comparación con los obtenidos por DHS. Para este cálculo se utilizó el test de Mann-Whitney con un 95% de certeza. El resultado de este test mostró que DHClustR construye jerarquías con una calidad estadísticamente significativa respecto a los construidos por el algoritmo DHS.

4.4.3. Número de niveles y grupos en la jerarquía

En este experimento se comparan los algoritmos de acuerdo al número de niveles y de grupos, que tienen las jerarquías con las logran su mejor valor de calidad, para cada colección de prueba (ver subsección 4.4.2).

Como se puede observar en la Tabla 7, el algoritmo propuesto construye jerarquías que tienen menos niveles y grupos, que las jerarquías formadas por el algoritmo DHS. Estas características son importantes para aplicaciones de recuperación y organización de información sobre la WWW [55,31,56]. En este tipo de aplicaciones los resultados son organizados en una jerarquía y los usuarios tienen que navegar por ella para encontrar la información que necesitan. Aunque el número de niveles y grupos de una jerarquía depende de las características intrínsecas de los datos utilizados, construir jerarquías con muchos niveles y grupos dificulta la búsqueda de información por parte de los usuarios.

Finalmente, es importante comentar que durante los experimentos de la subsección 4.4.2, el algoritmo DHS siempre fue el que construyó las jerarquías con más cantidad de niveles y grupos, incluso para los mismos valores de β que el algoritmo DHClustR.

Tabla 7. Número de niveles y grupos de la jerarquía con mejor valor de calidad, que forma cada algoritmo para las colecciones de prueba.

	Algoritmos			
	DHS		DHClustR	
Colección	#niveles	#grupos	#niveles	#grupos
AFP	5	502	2	134
Reu-Te	7	2525	2	190
Reu-Tr	8	5408	3	378
Reuter	9	7680	3	564
TDT	8	11644	3	2576
TDT-1	8	6383	3	2007
TDT-2	8	5819	3	1448
TDT-3	8	7345	3	1911
TDT-4	8	7463	3	1896
TDT-5	8	8535	3	2028
cacm	5	359	2	195
cisi	6	801	3	142

4.4.4. Eficiencia

Como se demostró en la subsección 4.3, DHClustR tiene una menor complejidad computacional que el algoritmo DHS y por lo tanto, es capaz de procesar una colección en menos tiempo de lo que tardaría DHS en procesar la misma colección. En esta subsección se realizaron algunos experimentos para mostrar la ventaja que le ofrece a DHClustR, tener esta diferencia de complejidad computacional respecto a DHS.

El primer experimento se enfocó en comparar los algoritmos, de acuerdo al tiempo empleado en procesar múltiples adiciones sobre la colección TDT. Cada curva de la Figura 5 representa el tiempo que emplea cada algoritmo para agrupar subcolecciones de tamaño 2000, 4000, 6000, …, 16 000; en el caso del algoritmo DHClustR se muestra el tiempo promedio de veinte ejecuciones sobre TDT, variando el orden de los documentos.

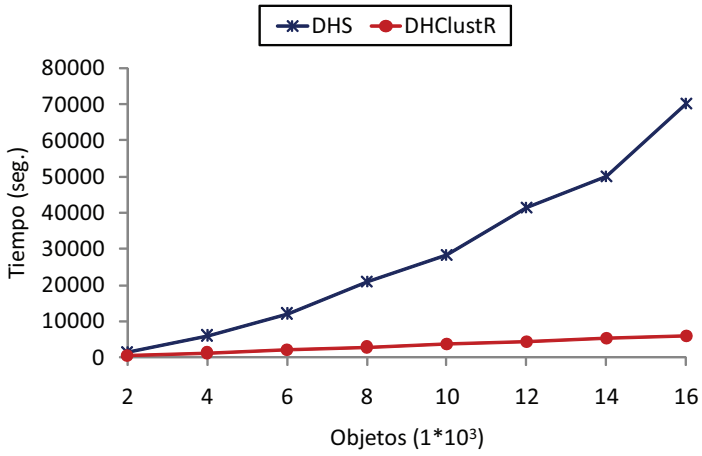


Fig. 5. Tiempo empleado por DHS y DHClustR en el procesamiento de múltiples adiciones sobre TDT.

Como era de esperar, el algoritmo propuesto emplea mucho menos tiempo que DHS, para procesar multiples adiciones sobre la colección TDT. Este experimento se repitió, variando la colección y el número de objetos que son adicionados, observándose siempre el mismo comportamiento.

El segundo experimento realizado se enfocó en comparar el tiempo que cada algoritmo emplea en procesar completamente algunas de las colecciones de prueba. En la Figura 6 se muestra el tiempo empleado por ambos algoritmos, en el procesamiento de las colecciones cisi, AFP, Reu-Te, Reu-Tr, Reuter y TDT-2, usando $\beta = 0{,}30$ para los dos algoritmos.

Como se puede observar en la Figura 6, el tiempo que emplea DHClustR para procesar cada colección es mucho menor que el utilizado por el algoritmo DHS. De forma análoga al experimento anterior, este experimento se repitió utilizando otras colecciones y otros valores de β, observándose siempre el mismo comportamiento.

4.5. Recapitulación

En esta sección se introdujo un nuevo algoritmo de agrupamiento jerárquico y dinámico, denominado DHClustR, que es capaz de formar jerarquías con traslape. DHClustR construye una jerarquía de grupos con traslape a través de la aplicación sucesiva del algoritmo DClustR, introducido también en este trabajo. Adicionalmente, DHClustR introduce un nuevo modelo para representar los grupos en la jerarquía, así como una nueva función para calcular la semejanza entre los grupos. Estos aspectos le permiten a DHClustR reducir las limitaciones del algoritmo DHS, con una complejidad computacional menor que la de DHS. El algoritmo DHS es el único algoritmo jerárquico dinámico que es capaz de construir jerarquías de grupos con traslape.

El algoritmo propuesto se comparó con el algoritmo DHS en varias colecciones estándares. Los experimentos realizados se enfocaron en comparar los algoritmos en cuanto a la calidad de las jerarquías formadas, el número de niveles y grupos de las jerarquías construidas, así como la eficiencia en el procesamiento de las colecciones. A partir de estos experimentos se puede concluir que el algoritmo DHClustR construye jerarquías con una mejor calidad que las construidas por el algoritmo DHS. Adicionalmente, DHClustR forma jerarquías con menos niveles y grupos que las formadas por DHS. Estas características hacen a DHClustR útil para aplicaciones como la organización de información sobre la WWW, donde construir jerarquías fáciles de navegar es importante [55,31,56]. Estos experimentos también mostraron que la estrategia de actualización de la jerarquía que propone DHClustR es claramente más rápida que la que emplea el algoritmo DHS.

A partir de lo anterior, se puede concluir que DHClustR es una mejor opción para enfrentar el problema del agrupamiento jerárquico traslapado de objetos, en un contexto dinámico, que los algoritmos del estado-del-arte.

5. Conclusiones y trabajo futuro

El desarrollo de algoritmos de agrupamiento jerárquicos y no jerárquicos continúa siendo objeto de interés debido a su amplia variedad de aplicaciones. Existen varios tipos de aplicaciones donde es común que los objetos puedan pertenecer a varios grupos; no obstante, la mayoría de los algoritmos de agrupamiento reportados en la literatura construyen grupos disjuntos. Adicionalmente, los algoritmos jerárquicos y no jerárquicos, capaces de formar grupos con traslape, que se han propuesto hasta el momento, tienen un conjunto de limitaciones que pueden reducir su utilidad en ciertos problemas prácticos.

En este trabajo se propusieron dos nuevos algoritmos de agrupamiento con traslape, DClustR y DHClustR, que reducen las limitaciones de los algoritmos relacionados del estado-del-arte y que además permiten actualizar eficientemente el agrupamiento formado cuando ocurren múltiples cambios en la colección.

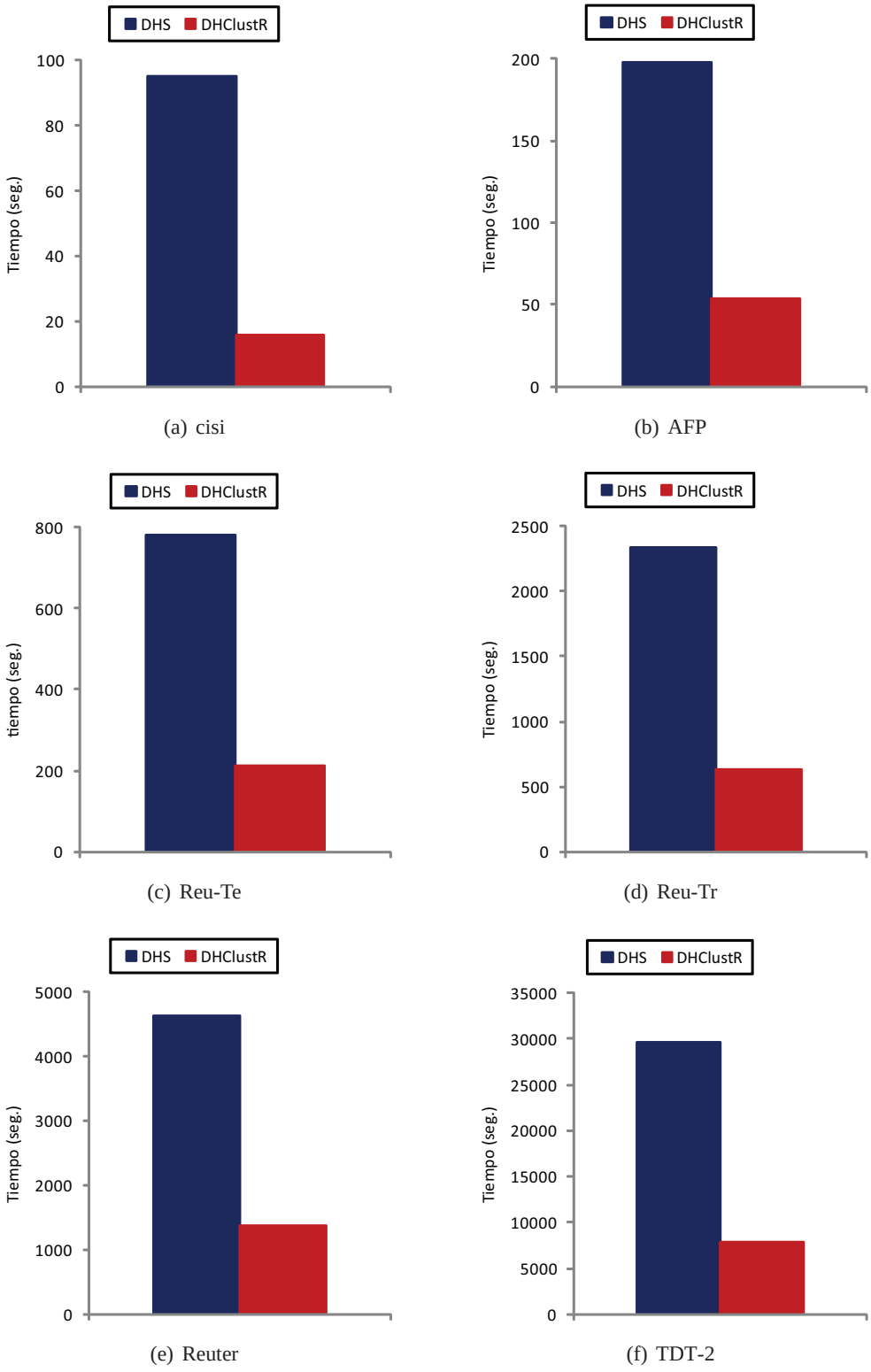


Fig. 6. Tiempo empleado por DHClustR y DHS para procesar las colecciones, usando β=0.30.

Se realizaron varios experimentos, utilizando varias colecciones estándares, en los que se evaluó el comportamiento de los algoritmos propuestos. Los experimentos se enfocaron en comparar los algoritmos propuestos y los algoritmos del estado del arte, atendiendo a varios aspectos de interés. Los experimentos con los algoritmos no jerárquicos estuvieron enfocados en comparar la calidad, cantidad y traslape de los grupos formados, así como el tiempo que tarda cada algoritmo en procesar múltiples adiciones y/o eliminaciones de objetos. Por otra parte, los experimentos con los algoritmos jerárquicos se enfocaron en comparar la calidad, número de niveles y número de grupos de la jerarquía formada por cada algoritmo, así como la eficiencia en el procesamiento de las colecciones.

A partir de los experimentos con los algoritmos no jerárquicos, se puede concluir que el algoritmo DClustR:

- Forma agrupamientos que tienen una calidad significativamente superior a los formados por los algoritmos no jerárquicos del estado-del-arte.
- Construye agrupamientos que tienen un menor número de grupos y un menor traslape que los formados por los algoritmos no jerárquicos relacionados.

De los experimentos con múltiples adiciones se obtienen dos conclusiones importantes:

a) La estrategia propuesta por DClustR para la actualización del agrupamiento es más rápida que la empleada por la mayoría de los algoritmos evaluados.
b) Aunque el algoritmo ISC es más rápido que DClustR, este último supera a ISC en cuanto a la calidad, número y traslape de los grupos formados; por lo tanto, el algoritmo DClustR ofrece una mejor relación eficacia-eficiencia que ISC.

Por otra parte, los experimentos con múltiples eliminaciones y modificaciones mostraron que el algoritmo propuesto es más rápido que los algoritmos dinámicos del estado-del-arte.

A partir de los experimentos con los algoritmos jerárquicos, se puede concluir que el algoritmo DHClustR:

- Construye jerarquías de grupos con traslape con una calidad significativamente superior a la calidad de las jerarquías formadas por el algoritmo DHS; este algoritmo es el único algoritmo jerárquico y dinámico, que forma jerarquías traslapadas.
- Construye jerarquías que tienen un menor número de niveles y grupos, que las formadas por DHS.

Por último, a partir de estos experimentos se puede concluir también que DHClustR es más rápido que el algoritmo DHS.

Con base en lo mencionado anteriormente, se puede concluir que DHClustR y DClustR representan mejores opciones para enfrentar el problema del agrupamiento con traslape en un contexto dinámico, tanto jerárquico como no jerárquico, que los algoritmos existentes en el estado-del-arte.

Como trabajo futuro se piensa extender estos resultados en el contexto del agrupamiento conceptual, de forma que además de los grupos, se cuente con una descripción intensional de los mismos. Adicionalmente, se pretende desarrollar versiones paralelas de los algoritmos, con el objetivo de aumentar la aplicabilidad de los mismos.

Referencias bibliográficas

1. Amigó, E., Gonzalo, J., Artiles, J., Verdejo, F.: A comparison of extrinsic clustering evaluation metrics based on formal constraints. Information Retrieval **12** (2009) 461–486

2. Bae, E., Bailey, J., Dong, G.: A clustering comparison measure using density profiles and its application to the discovery of alternate clusterings. Data Mining and Knowledge Discovery **21**(3) (2010) 427–471
3. Pfitzner, D., Leibbrandt, R., Powers, D.: Characterization and evaluation of similarity measures for pairs of clusterings. Knowledge and Information Systems **19**(3) (2009) 361–394
4. Martínez-Trinidad, J., Guzmán-Arenas, A.: The logical combinatorial approach to pattern recognition, an overview through selected works. Pattern Recognition **34**(4) (2001) 741–751
5. Jain, A., Murty, M.N., Flynn, P.: Data clustering: A review. ACM Computing Surveys **31**(3) (1999) 264–323
6. Omran, M., Engelbrecht, A., Salman, A.: An overview of clustering methods. Intelligent Data Analysis **11**(6) (2007) 583–605
7. Banerjee, A., Krumpelman, C., Basu, S., Mooney, R., Ghosh, J.: Model-based overlapping clustering. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (KDD2005). (2005) 532–537
8. Cleuziou, G.: A generalization of k-means for overlapping clustering. Technical Report RR-2007-15, Université d Orléans, LIFO (2007)
9. Cleuziou, G.: Two variants of the okm for overlapping clustering. In: Advances in Knowledge Discovery and Management: Studies in Computational Intelligence. (2010) 149–166
10. Cleuziou, G., Martin, L., Vrain, C.: Poboc: an overlapping clustering algorithm. application to rule-based classification and textual data. In: Proceedings of the 16th European Conference on Artificial Intelligence ECAI-04. (2004) 440–444
11. Fu, Q., Banerjee, A.: Multiplicative mixture models for overlapping clustering. In: Proceedings of the IEEE International Conference on Data Mining. (2008) 791–796
12. Gago-Alonso, A., Pérez-Suárez, A., Medina-Pagola, J.: Acons: a new algorithm for clustering documents. In: Proceedings of the 12th Iberoamerican Congress on Pattern Recognition (CIARP2007), LNCS 4756. (2007) 664–673
13. Gil-García, R., Badía-Contelles, J., Pons-Porrata, A.: Extended star clustering algorithm. In: Proceedings of the 8th Iberoamerican Congress on Pattern Recognition (CIARP2003), LNCS 2905. (2003) 480–487
14. Gil-García, R., Badía-Contelles, J., Pons-Porrata, A.: Parallel algorithm for extended star clustering. In: Proceedings of the 9th Iberoamerican Congress on Pattern Recognition (CIARP2004), LNCS 3287. (2004) 402–409
15. Pérez-Suárez, A., Medina-Pagola, J.: 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) 248–262
16. Baumes, J., Goldberg, M., Krishnamoorty, M., Magdon-Ismail, M., Preston, N.: Finding communities by clustering a graph into overlapping subgraphs. In: Proceedings of IADIS Applied Computing. (2005) 97–104
17. Baumes, J., Goldberg, M., Magdon-Ismail, M.: Efficient identification of overlapping communities. In: Proceedings of the IEEE International Conference on Intelligence and Security Informatics (ISI), LNCS 3495. (2005) 27–36
18. Davis, G., Carley, K.: Clearing the fog: Fuzzy, overlapping groups for social networks. Social Networks **30**(3) (2008) 201–212
19. Goldberg, M., Kelley, S., Magdon-Ismail, M., Mertsalov, K., Wallace, A.: Finding overlapping communities in social networks. In: Proceedings of the IEEE Second International Conference on Social Computing (SocialCom2010). (2010) 104–113
20. Gregory, S.: An algorithm to find overlapping community structure in networks. In: Proceedings of the PKDD 2007, LNAI 4702. (2007) 91–102
21. Gregory, S.: A fast algorithm to find overlapping communities in networks. In: Proceedings of the 12th ECML KDD, LNAI 5212. (2008) 408–423
22. Heller, K., Ghahramani, Z.: A nonparametric bayesian approach to modeling overlapping clusters. In: Proceedings of the Eleventh International Conference on Artifical Intelligence and Statistics (AISTATS). (2007) 187–194
23. Macropol, K., Can, T., Singh, A.: Rrw: repeated random walks on genome-scale protein networks forlocal cluster discovery. BMC Bioinformatics **10**(283) (2009)
24. Magdon-Ismail, M., Purnell, J.: Ssde-cluster: Fast overlapping clustering of networks using sampled spectral distance embedding and gmms. In: Proceedings of the 3rd IEEE International Conference on Social Computing (SocialCom2011). (2011) 756–759
25. Palla, G., Derényi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. Nature **435**(7043) (2005) 814–824
26. Sap, M., Mohebi, E.: Hybrid self organizing map for overlapping clusters. International Journal of Signal Processing, Image Processing and Pattern Recognition **1** (2008) 11–20
27. Zhang, S., Wang, R., Zhang, X.: Identification of overlapping community structure in complex networks using fuzzy c-means clustering. Physica A: Statistical Mechanics and its Applications **374**(1) (2007) 483–490
28. Aslam, J., Pelekhov, K., Rus, D.: Static and dynamic information organization with star clusters. In: Proceedings of the seventh international conference on Information and knowledge management. (1998) 208–217
29. Zamir, O., Etziony, O.: Web document clustering: A feasibility demonstration. In: Proceedings of the 21st Annual International ACM SIGIR Conference. (1998) 46–54

30. Pons-Porrata, A., Ruiz-Shulcloper, J., Berlanga-Llavorí, R., Santiesteban-Alganza, Y.: Un algoritmo incremental para la obtención de cubrimientos con datos mezclados. Reconocimiento de Patrones. Avances y Perspectivas. Research on Computing Science, CIARP2002 (2002) 405–416
31. Hammouda, K., Kamel, M.: Efficient phrase-based document indexing for web document clustering. IEEE Transactions on Knowledge and Data Engineering **16**(10) (2004) 1279–1296
32. Pérez-Suárez, A., Martínez-Trinidad, J., Carrasco-Ochoa, J., Medina-Pagola, J.: A new incremental algorithm for overlapped clustering. In: Proceedings of the 14th Iberoamerican Congress on Pattern Recognition (CIARP2009), LNCS 5856. (2009) 497–504
33. Pérez-Suárez, A., Martínez-Trinidad, J., Carrasco-Ochoa, J., Medina-Pagola, J.: A dynamic clustering algorithm for building overlapping clusters. Intelligent Data Analysis **16**(2) (2012) 211–232
34. Gil-García, R., Pons-Porrata, A.: Dynamic hierarchical algorithms for document clustering. Pattern Recognition Letters **31**(6) (2010) 469–477
35. AbellaPérez, R., MedinaPagola, J.: An incremental text segmentation by clustering cohesion. In: Proceedings of HaCDAIS 2010. (2010) 65–72
36. Zhao, Y., Karypis, G.: Evaluation of hierarchical clustering algorithms for document datasets. In: Proceedings of the eleventh international conference on Information and knowledge management. (2002) 515–524
37. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Commun. ACM **18**(11) (1975) 613–620
38. Berry, M.: Survey of Text Mining, Clustering, Classification and Retrieval. Springer-Verlag (2004)
39. Knuth, D.E.: The Art of Computer Programming. Volume 3. Addison-Wesley (1973)
40. Greengrass, E.: Information retrieval: A survey. Technical Report TR-R52-008-001 (2001)
41. Zhao, Y., Karypis, G.: Criterion functions for document clustering: Experiments and analysis. Technical Report 01–40, Department of Computer Science, University of Minnesota, Minneapolis, MN (2001)
42. Larsen, B., Aone, C.: Fast and effective text mining using linear-time document clustering. Knowledge Discovery and Data Mining (1999) 16–22
43. Halkidi, M., Batistakis, Y., Vazirgiannis, M.: On clustering validation techniques. Journal of Intelligent Information Systems **17**(2-3) (2001) 107–145
44. Meila, M.: Comparing clusterings by the variation of information. In: Proceedings of COLT/Kernel 2003. (2003) 173–187
45. Steinbach, M., Karypis, G., Kumar, V.: A comparison of document clustering techniques. In: Proceedings of KDD 2000. (2000)
46. Bakus, J., Hussin, M., Kamel, M.: A som-based document clustering using phrases. In: Proceedings of ICONIP'02. (2002) 2212–2216
47. Rosenberg, A., Hirschberg, J.: V-measure: A conditional entropy-based external cluster evaluation measure. In: Proceedings of EMNLP-CoNLL 2007. (2007) 410–420
48. Ramírez, E.H., Brena, R., Magatti, D., Stella, F.: Topic model validation. Neurocomputing **76** (2012) 125–133
49. Bagga, A., Baldwin, B.: Entity-based cross-document coreferencing using the vector space model. In: Proceedings of COLING-ACL'98. (1998) 79–85
50. Gil-García, R.J.: Algoritmos de agrupamiento sobre grafos y su paralelización. PhD thesis, Departamento de Ingenieria y Ciencia de los Computadores. Universidad Jaume I. España (2005)
51. Gil-García, R., Badía-Contelles, J., Pons-Porrata, A.: Dynamic hierarchical compact clustering algorithm. In: Proceedings of the X Iberoamerican Congress on Pattern Recognition (CIARP2005), LNCS 3773, Springer-Verlag Berlin Heidelberg (2005) 302–310
52. Allan, J., Feng, A., Bolivar, A.: Flexible intrinsic evaluation of hierarchical clustering for tdt. In: Proceedings of the Twelfth ACM International Conference on Information and Knowledge Management (CIKM 2003). (2003) 263–270
53. James, A., ed.: Topic detection and tracking: event-based information organization. Kluwer Academic Publishers, Norwell, MA, USA (2002)
54. Gil-García, R., Pons-Porrata, A.: Hierarchical star clustering algorithm for dynamic document collections. In: Proceedings of the 13th Iberoamerican congress on Pattern Recognition: Progress in Pattern Recognition, Image Analysis and Applications. (2008) 187–194
55. Beil, F., Ester, M., Xu, X.: Frequent term-based text clustering. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. KDD '02, New York, NY, USA, ACM (2002) 436–442
56. Fung, B., Wang, K., Esther, M.: Hierarchical document clustering using frequent itemsets. In: Proceedings of the Third SIAM International Conference on Data Mining. (2003) 59–70

RT_019, marzo 2013
Aprobado por el Consejo Científico CENATAV

Editor: Lic. Lucía González Bayona
Diseño de Portada: Di. Alejandro Pérez Abraham
RNPS No. 2143
ISSN 2072-6260
Indicaciones para los Autores:
Seguir la plantilla que aparece en www.cenatav.co.cu
C E N A T A V
7ma. A No. 21406 e/214 y 216, Rpto. Siboney, Playa;
La Habana. Cuba. C.P. 12200
Impreso en Cuba