Algoritmos dinámicos para el agrupamiento con traslape

Por:

Airel Pérez Suárez

Tesis sometida como requisito parcial para obtener el grado de:
Doctor en Ciencias en el Area de Ciencias Computacionales en el
Instituto Nacional de Astrofísica, Optica y Electrónica.

Supervisada por:

Dr. José Francisco Martínez Trinidad
Dr. José Eladio Medina Pagola

Comité revisor:

Dra. Claudia Feregrino Uribe (INAOE)
Dr. Jesús Antonio González Bernal (INAOE)
Dr. Saúl E. Pomares Hernández (INAOE)
Dr. Leopoldo Altamirano Robles (INAOE)
Dr. David Eduardo Pinto Avendaño (BUAP)

Luis Enrique Erro # 1, Tonantzintla,
Puebla, 72840, México
21 de julio de 2011

©INAOE Enero 2011
Derechos Reservados
El autor otorga al INAOE el permiso de reproducir y distribuir copias de esta tesis en su totalidad o en partes mencionando la fuente.

Resumen

El *agrupamiento* es una técnica del Aprendizaje Automático y de la Minería de Datos, que ha sido utilizada en varias áreas como la medicina, el marketing, el análisis de redes sociales y la bioinformática, entre otras. A pesar de los resultados que se han alcanzado hasta el momento en el estudio y desarrollo de nuevos algoritmos de agrupamiento, todavía existen algunas limitaciones en los mismos, que son solucionadas en el marco de esta investigación doctoral.

La mayoría de los algoritmos de agrupamiento no permiten formar grupos con traslape. Sin embargo, existen varias aplicaciones como la detección de tópicos, la segmentación de documentos, la organización de información y el análisis de noticias, entre otras, donde los objetos pueden pertenecer a más de un grupo; este tipo de aplicaciones necesitan de algoritmos de agrupamiento que permitan formar grupos con traslape. Los algoritmos traslapados que se han propuesto hasta el momento, tienen un conjunto de limitaciones que pueden reducir su utilidad en ciertos problemas prácticos. Estas limitaciones están relacionadas principalmente con: (a) la necesidad de ajustar varios parámetros cuyos valores dependen de la colección a agrupar, (b) la construcción de un gran número de grupos, generalmente con un bajo promedio de elementos por grupo y (c) la obtención de agrupamientos con un alto nivel de traslape. Adicionalmente, la mayoría de los algoritmos de agrupamiento traslapado son incapaces de satisfacer nuevos requerimientos tales como: (i) la necesidad de actualizar el agrupamiento previamente construido, cuando cambia la colección y (ii) la necesidad de crear estructuras jerárquicas, en las cuales sea permitido el traslape entre los grupos de un mismo nivel.

En este trabajo de investigación doctoral se introducen dos nuevos algoritmos de agrupamiento traslapado, DClustR y DHClustR, que abordan los requerimientos anteriormente comentados y que además, solucionan las limitaciones a), b) y c). DClustR es un algoritmo dinámico no jerárquico, que se basa en conceptos de Teoría de grafos para formar un conjunto de grupos con traslape. DClustR introduce una nueva estrategia para la formación del agrupamiento, así como una nueva estrategia para la actualización de este conjunto de grupos, cuando ocurren múltiples adiciones, eliminaciones y modificaciones de objetos de la colección. Por otra parte, DHClustR es un algoritmo dinámico, jerárquico y aglomerativo, que construye una jerarquía de grupos traslapados a través de la aplicación sucesiva del algoritmo DClustR. Para construir el agrupamiento del primer nivel de la jerarquía, se aplica el algoritmo DClustR a los objetos de la colección; a partir de este punto, los objetos a agrupar en cada nivel son los grupos formados en el nivel inmediato inferior. DHClustR introduce además una estrategia para la actualización de la jerarquía formada, cuando ocurren múltiples adiciones, eliminaciones y modificaciones de objetos de la colección.

Como parte de la investigación desarrollada se realizaron varios experimentos, utilizando varias colecciones estándares de datos, en los que se evaluó el comportamiento de los algoritmos propuestos. Los experimentos realizados mostraron que los algoritmos propuestos forman agrupamientos con una calidad significativamente superior a los que forman los algoritmos del estado del arte. Con base en estos resultados, se puede concluir que DHClustR y DClustR constituyen 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.

Abstract

Clustering is a Data Mining and Machine Learning technique that has been used in several areas like medicine, marketing, social network analysis and bioinformatics, among others. Although, several clustering algorithms have been proposed, they have some drawbacks that are solved through this PhD research.

Most clustering algorithms do not allow building overlapping clusterings. However, there are several applications like topics detection, document segmentation, information organization and news analysis, among others, where it is common for objects to belong to more than one cluster; these applications need clustering algorithms able to build overlapping clusters. The majority of the current overlapping clustering algorithms have some drawbacks which can reduce their usefulness in practical applications. These limitations are mainly related with: (a) the necessity of tuning several parameters whose values depend on the collection to cluster, (b) the production of a large number of clusters, usually with a low average of elements per cluster, and (c) the production of clusters with high overlapping. Besides, most of overlapping clustering algorithms are unable to satisfy new requirements such as: (i) the necessity of updating the clustering when the collection changes and (ii) the necessity of building hierarchies of clusters, in which the overlapping among the clusters of the same level is allowed.

This PhD research introduces two new overlapping clustering algorithms, DClustR and DHClustR, both satisfy the requirements above mentioned and solve the limitations a), b) and c). DClustR is a dynamic and non hierarchical algorithm, based on concepts of graph theory, which builds a set of overlapping clusters. DClustR introduces a new strategy for building the clustering

and also it introduces a new strategy for updating this clustering when the collection changes, due to multiple additions, eliminations or modifications of objects. On the other hand, DHClustR is a dynamic agglomerative hierarchical clustering algorithm which builds a hierarchy of overlapping clusters, using the DClustR algorithm for building the clustering of each level. For building the clustering of the first level, DClustR is applied over the collection of objects; from this point on, the objects to be clustered at each level are the clusters of the previous level. DHClustR introduces also a new strategy for updating the hierarchy when the collection changes, due to multiple additions, eliminations or modifications of objects.

In order to evaluate the proposed algorithms, a set of experiments were done over several standard data collections. The experiments show that the proposed algorithms build clusters having a quality significantly greater, from a statistically point of view, than the one of the clusters built by the algorithms of the state of the art. Based on these results, it can be concluded that DHClustR and DClustR are better options for solving hierarchical and non hierarchical problems of overlapping clustering in dynamic environments, than the existing algorithms of the state of the art.

A mis abuelos Tata y Cheo,
a mi abuela Cristina

Agradecimientos

Quiero expresar mis más sinceros agradecimientos a mis directores de tesis Dr. Francisco Martínez Trinidad y Dr. José Eladio Medina Pagola, por el asesoramiento y los consejos brindados durante el desarrollo de esta investigación. Sus enseñanzas han sido fundamentales para el desarrollo de esta tesis.

Quiero agradecer al Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV), Habana, Cuba, en especifico a su director Dr. José Ruiz Shulclopler, por el apoyo que ha dado a la formación de nuevos doctores en áreas de la Minería de Datos y el Reconocimiento de Patrones.

Agradezco a los miembros de mi comité por sus revisiones y sugerencias transmitidas a lo largo de esta investigación. Gracias a Dra. Claudia Feregrino Uribe, Dr. Leopoldo Altamirano Robles, Dr. Jesús A. González Bernal, Dr. Saúl Pomares Hernández y Dr. David Eduardo Pinto Avendaño.

Agradezco al Dr. Jesús Ariel Carrasco Ochoa por contribuir positivamente, con sus consejos y observaciones oportunas, en el desarrollo de esta investigación y en específico, en mi formación como investigador. De igual forma quiero agradecerle la ayuda brindada desde mi llegada a México.

Quiero dedicar un agradecimiento muy especial a mis padres, Rafael y Mercedes, por todo el amor y apoyo incondicional brindado desde pequeño. La perseverancia que he observado siempre en ustedes me ha servido como impulso para superar innumerables obstáculos en mi vida y, específicamente, en esta investigación.

Agradezco profundamente a mi hermano Arian por todo el cariño que me

tiene y por la confianza que siempre ha depositado en mí. Gracias por estar ahí cada vez que te necesito.

Quiero agradecer también a mi esposa Yaraidys por su amor incondicional. Gracias a tu comprensión he podido dedicar tiempo para el desarrollo de esta investigación y gracias al ánimo que me das he podido encarar mis problemas con una sonrisa.

Agradezco también a mi abuela mima, a mis tíos, a mi tía, a mi cuñada, al chenchin; al resto de la familia.

Quiero agradecer a Raudel por ser mi mejor amigo y por acompañarme en las malas y en las buenas. Aunque me desagrada que siempre me critiques, tengo que aceptar que gracias a tus críticas he podido reducir la cantidad de veces en que me equivoco, las cuales, aclaro, no son pocas. De igual forma, quiero agradecerle a Chang por la amistad que me ha brindado desde que lo conocí y por las tantas veces que me ha ayudado.

Agradezco a los amigos del barrio, específicamente a Abraham y a Osmany. Gracias por su amistad y por estar siempre dispuestos a ayudar.

Agradezco a los trabajadores del CENATAV, es específico a mis compañeros del departamento de MD. Gracias a Gail, Sandro, Niusvel, Kadir, Rainier, Reynel, Edel, Gago, Raúl y Alfredo. No me imagino otro lugar dentro del CENATAV en el cual me sienta tan a gusto trabajando. Agradezco al colectivo de ICT por siempre estar dispuestos a ayudarme. Agradezco además a Arturo por el tiempo y esfuerzo dedicados en la revisión de los artículos derivados de esta investigación.

Agradezco al Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) por brindarme un espacio para desarrollar mis estudios. Agradezco también al CONACyT por el financiamiento otorgado para el desarrollo de esta investigación (proyecto CB-2008-01-106443 y beca doctoral 32040).

Sé que puedo estar omitiendo el nombre de algunas personas, tanto de México como de Cuba, sin las cuales esta investigación no hubiera sido posible; personas que me han brindado su amistad y apoyo. Mis disculpas a estas personas y sepan que, aún cuando no las haya mencionado, las tengo presente y estoy consciente de su ayuda.

Índice general

Capítulo 1

Introducción

1.1. Introducción

Desde hace varios años se han reportado avances en las tecnologías para el manejo de almacenes de datos (*data warehousing*) y en los sistemas de bases de datos. El desarrollo alcanzado ha permitido transformar el gran volumen de datos que se genera diariamente (imágenes, texto, video, etc.), en información de enorme valor para aplicaciones como las finanzas, la atención a la salud, el marketing, la ingeniería y la bioinformática, entre otras. Para un usuario de estas aplicaciones, resulta de vital importancia extraer, de los datos almacenados, algún conocimiento útil que lo ayude en la toma de decisiones. Una de las técnicas que se ha usado frecuentemente para este propósito es el *agrupamiento*.

El *agrupamiento* es una de las técnicas fundamentales en el Aprendizaje Automático y en la Minería de Datos (Bae *et al.*, 2010). 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 (Pfitzner *et al.*, 2009). De forma general, se asume que los objetos a agrupar están descritos por m rasgos o variables, cuyos valores pueden ser cuantitativos o cualitativos. Adicional-

mente, existen problemas en los que se puede desconocer el valor para algunos de los rasgos (Martínez-Trinidad and Guzmán-Arenas, 2001).

El conjunto de grupos obtenido por un algoritmo de agrupamiento puede ser disjunto, traslapado o difuso (Jain and Dubes, 1988). 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 matemática, física, química, política, religión y teología.

Los algoritmos de agrupamiento pueden ser clasificados atendiendo a diferentes criterios (Jain *et al.*, 1999). Uno de estos criterios divide los algoritmos de agrupamiento en *restringidos* o *libres*, de acuerdo a si requieren, o no, conocer a priori el número de grupos a obtener. En la mayoría de los problemas prácticos, el número de grupos a formar es desconocido. Por lo tanto, esta investigación se centrará en los algoritmos de agrupamiento libres.

En la literatura se encuentran reportados diversos algoritmos de agrupamiento. No obstante, el desarrollo de nuevos algoritmos de agrupamiento continúa siendo objeto de interés debido a su amplia variedad de aplicaciones (Omran *et al.*, 2007). Existen varios ejemplos de la aplicación de técnicas de agrupamiento para la solución de problemas prácticos en diferentes areas como: Medicina (Schmidt *et al.*, 2010; Mahata, 2010), Marketing (Jiang and Tuzhilin, 2009), Detección de fraudes (Palshikar and Apte, 2008), Clasificación de imágenes (Chang *et al.*, 2010), Análisis de redes sociales (Narasimhamurthy *et al.*, 2010), Detección de intrusos (Liu *et al.*, 2004), Identificación de duplicados (Costa *et al.*, 2010) y Bioinformática (Kianmehr *et al.*, 2010; Gupta *et al.*, 2010), entre otras.

A pesar de los resultados que se han alcanzado hasta el momento en el estudio y desarrollo de nuevos algoritmos de agrupamiento, todavía existen algunas limitaciones en los mismos, que serán tratadas en el marco de esta investigación doctoral.

1.2. Descripción del problema

La mayoría de los algoritmos de agrupamiento reportados en la literatura construyen grupos disjuntos. Sin embargo, existen varios tipos de aplicaciones donde es común que los objetos puedan pertenecer a varios grupos, por ejemplo: la detección de tópicos (Aslam *et al.*, 1998), la segmentación (Abella-Pérez and Medina-Pagola, 2010), la organización de documentos web (Hammouda and Kamel, 2004) y el análisis de flujos de noticias (Pons-Porrata *et al.*, 2002), entre otros. En lo siguiente, los algoritmos de agrupamiento que permitan formar grupos traslapados serán referidos como algoritmos traslapados.

En la literatura se han reportado varios algoritmos de agrupamiento que permiten obtener grupos traslapados (Aslam *et al.*, 1998; Zamir and Etziony, 1998; Pons-Porrata *et al.*, 2002; Gil-García *et al.*, 2003; Hammouda and Kamel, 2004; Pérez-Suárez and Medina-Pagola, 2007; Gago-Alonso *et al.*, 2007; Pérez-Suárez *et al.*, 2009); no obstante, estos algoritmos presentan algunas limitaciones que pueden reducir su utilidad en ciertos problemas prácticos. Estas limitaciones están relacionadas principalmente con: (a) la necesidad de ajustar varios parámetros cuyos valores dependen de la colección a agrupar, (b) la construcción de un gran número de grupos, generalmente con un bajo promedio de elementos por grupo y (c) la obtención de agrupamientos con un alto nivel de traslape.

La limitación (a) dificulta la aplicación de un algoritmo de agrupamiento en un problema real, pues en la práctica los usuarios desconocen cómo ajustar los parámetros de los algoritmos a las características de los datos que desean agrupar. Por otra parte, la limitación (b) puede reducir la utilidad de aplicar un algoritmo de agrupamiento en aplicaciones que procesen colecciones de gran tamaño. Por lo general, el número correcto de grupos es desconocido en un problema real y es el experto quien determina si obtener más o menos grupos es bueno o malo. No obstante, cuando se quiere descubrir relaciones que existen entre los objetos de una colección, generalmente se espera obtener un conjunto de grupos que sea razonablemente pequeño en comparación con el tamaño de la colección. Es importante notar que, cuando el número de grupos obtenidos es muy grande, como ocurre con la mayoría de los algoritmos traslapados del estado del arte, analizar todos los grupos se vuelve tan complejo como analizar la colección.

Por otra parte, aunque es importante la obtención de grupos con traslape, hay aplicaciones en las que la limitación (c) puede dificultar el análisis de los resultados, así como la obtención de conclusiones útiles a partir de los datos. En el contexto de la segmentación de documentos por tópicos, un elevado traslape entre los tópicos determinados puede reflejar una mala segmentación y consecuentemente, incidir negativamente en el análisis de los resultados (Abella-Pérez and Medina-Pagola, 2010). Algo similar ocurre en aplicaciones que organizan los resultados de búsquedas sobre la web (Zamir and Etziony, 1998; Hammouda and Kamel, 2004).

Con el incremento en las capacidades de producir y almacenar información, así como con el desarrollo de diversos servicios sobre internet, han surgido nuevos requerimientos para los algoritmos de agrupamiento que permiten obtener grupos traslapados. Estos nuevos requerimientos están relacionados principalmente con: (i) la necesidad de procesar cambios que puedan ocurrir sobre la colección y actualizar el agrupamiento previamente construido y (ii) la necesidad de crear estructuras jerárquicas para agrupar a los objetos, en las cuales sea permitido el traslape entre los grupos de un mismo nivel. En lo siguiente, estas jerarquías serán nombradas *jerarquías traslapadas*.

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 de agrupamiento 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. Solo unos pocos de los algoritmos traslapados reportados, son capaces de procesar cambios en una colección que se encuentra agrupada; no obstante, la mayoría de estos algoritmos solo procesan adiciones.

De acuerdo al requerimiento (ii), los algoritmos de agrupamiento pueden ser clasificados en *jerárquicos* y *no jerárquicos*. Los algoritmos *jerárquicos* son aquellos que obtienen una jerarquía de grupos. En esta jerarquía, los grupos pertenecientes al nivel superior representan abstracciones o conceptos más generales que los del nivel inferior. Cada grupo de un nivel puede verse como la unión de varios grupos del nivel inmediato inferior. Por otra parte, los algoritmos *no jerárquicos* son aquellos que no construyen una jerarquía de grupos sino un único conjunto de grupos.

Los algoritmos jerárquicos pueden ser *aglomerativos* o *divisivos* (Zhao *et al.*, 2005). Los algoritmos aglomerativos parten de considerar a cada objeto de la colección como un grupo y construyen la jerarquía a partir de los grupos del nivel inmediato inferior, uniendo los dos grupos más semejantes. Este proceso de construcción de la jerarquía termina cuando todos los objetos pertenecen al mismo grupo. Algunos ejemplos de algoritmos aglomerativos son el Average-link, el Complete-link y el Single-link (Jain and Dubes, 1988). Por otra parte, los algoritmos divisivos parten de considerar que todos los objetos de la colección pertenecen a un mismo grupo y construyen la jerarquía a partir de los grupos del nivel inmediato superior, dividiendo en dos al grupo que cumpla una condición predefinida. Este proceso de construcción de la jerarquía termina cuando cada objeto forma un grupo independiente. Un ejemplo de un algoritmo divisivo es el Bisecting K-means (Zhao *et al.*, 2005).

Analizando los algoritmos de agrupamiento reportados en la literatura, se puede afirmar que aunque existen varios algoritmos jerárquicos reportados (Wai-chiu and Fu, 2000; Widyantoro *et al.*, 2002;

Pons-Porrata *et al.*, 2004; Chung and McLeod, 2005; Gurrutxaga *et al.*, 2009; Gil-García and Pons-Porrata, 2010), la mayoría de estos algoritmos no permiten obtener jerarquías traslapadas, no permiten actualizar la jerarquía formada cuando la colección cambia producto de adiciones, eliminaciones o modificaciones y además, presentan las limitaciones (a), (b) y (c) antes mencionadas.

Por lo expuesto anteriormente, el problema que se aborda en esta investigación doctoral es el desarrollo de algoritmos de agrupamiento, jerárquicos y no jerárquicos, que sean dinámicos y que permitan obtener grupos con traslape. Los algoritmos que se proponen en esta investigación doctoral están enfocados en resolver las limitaciones comentadas anteriormente.

1.3. Objetivos

Una clase muy importante dentro de los algoritmos de agrupamiento, lo constituyen los algoritmos basados en grafos. Este tipo de algoritmos construye un grafo de semejanzas con la colección de objetos a agrupar y posteriormente, forman el agrupamiento a partir del cubrimiento de este grafo o de un sub-grafo del mismo. Informalmente, un grafo de semejanza es el grafo completo (Aho *et al.*, 1983), en el cual los vértices son los objetos de la colección y las aristas se etiquetan con la semejanza entre los objetos que la conforman.

Una característica importante de los algoritmos basados en grafos, es que no imponen restricciones al espacio de representación de los objetos. Por otro lado, estos algoritmos tampoco restringen la medida de semejanza utilizada para la formación de las aristas. Estas características aumentan el campo de aplicación de dichos algoritmos.

Aplicando un razonamiento similar sobre los algoritmos jerárquicos que se encuentran reportados en la literatura, se puede comprobar que los que han sido más estudiados y desarrollados son los algoritmos aglomerativos (Zhao *et al.*, 2005). Estos algoritmos comienzan considerando a los objetos como grupos individuales y posteriormente, unen en cada iteración los grupos más semejantes; este proceso termina cuando todos los objetos pertenecen al mismo grupo. Existen trabajos que han mostrado que los algoritmos jerárquicos aglomerativos obtienen generalmente mejores

resultados que los divisivos (Gil-García *et al*., 2005; Puzicha *et al*., 2000). Adicionalmente, en los últimos años se han reportado varios algoritmos aglomerativos que han obtenido buenos resultados de eficacia y eficiencia (Gil-García and Pons-Porrata, 2008, 2010).

Con base en lo planteado anteriormente, la presente investigación doctoral tiene los siguientes objetivos:

Objetivo general

Desarrollar algoritmos de agrupamiento, jerárquicos y no jerárquicos, que sean dinámicos y que permitan construir grupos traslapados. Los algoritmos desarrollados deben alcanzar, respecto a los algoritmos reportados en la literatura, un rendimiento superior en cuanto a medidas de eficacia y un rendimiento similar o superior respecto a la eficiencia.

Objetivos específicos

1. Diseñar e implementar un algoritmo de agrupamiento, que sea incremental y permita obtener un conjunto de grupos con traslape.
2. Diseñar e implementar un algoritmo de agrupamiento, que sea dinámico y permita obtener un conjunto de grupos con traslape.
3. Diseñar e implementar un algoritmo de agrupamiento jerárquico aglomerativo, que sea incremental, utilice el algoritmo desarrollado en el objetivo específico 1 y permita obtener una jerarquía traslapada.
4. Diseñar e implementar un algoritmo de agrupamiento jerárquico aglomerativo, que sea dinámico, utilice el algoritmo desarrollado en el objetivo específico 2 y permita obtener una jerarquía traslapada.

1.4. Organización de la tesis

El contenido de este documento está organizado en 5 capítulos. En el capítulo 2, se describen los trabajos del estado del arte que se encuentran relacionados con la presente investigación. De cada algoritmo se describe cómo

se representan los objetos de la colección, cuál es el criterio para formar los grupos y qué limitaciones tiene.

En el capítulo 3, se introduce la estrategia utilizada por DClustR para la construcción de un conjunto de grupos traslapados como un nuevo algoritmo estático llamado OClustR. Posteriormente, se presenta la estrategia que 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 relacionados.

En el capítulo 4, se introduce la estrategia que utiliza DHClustR para la creación de la jerarquía de grupos como un nuevo algoritmo jerárquico estático llamado HClustR. A continuación, se presenta la estrategia que permite la actualización de la jerarquía, cuando la colección se modifica producto de múltiples adiciones y/o eliminaciones de objetos. Finalmente, se expone un conjunto de experimentos en los que se compara el comportamiento del algoritmo propuesto contra los algoritmos relacionados.

Finalmente, en el capítulo 5, se presentan las conclusiones, se mencionan las aportaciones obtenidas y se describen, tomando como base los resultados expuestos en este documento, algunas direcciones que se pueden seguir como trabajo futuro.

Capítulo 2

Trabajo relacionado

En la sección 2.1 se introducen los conceptos preliminares necesarios para comprender los algoritmos que forman parte del trabajo relacionado de la presente investigación. Posteriormente, en la sección 2.2 se hace un análisis crítico de los algoritmos de agrupamiento, tanto jerárquicos como no jerárquicos, que permiten formar grupos traslapados y procesar cambios en la colección. De cada algoritmo se describe cómo se representan los objetos de la colección, cuál es el criterio utilizado para formar los grupos y qué limitaciones tiene. Finalmente, en la sección 2.3 se presentan las conclusiones del capítulo.

2.1. Conceptos preliminares

Sea $O = \{o_1, o_2, \ldots, o_k\}$ una colección de objetos, $\beta \in [0, 1]$ un parámetro y $S(o_i, o_j)$ una función de semejanza tal que $\forall o_i, o_j \in O, o_i \neq o_j$, $S(o_i, o_j)$=$S(o_j, o_i)$.

Definición 2.1 (Grafo de β-semejanza)**.** *Un grafo de β-semejanza se denota por $G_\beta = \langle V, E_\beta \rangle$ y es el grafo no dirigido en el cual $V = O$ y $(o_i, o_j) \in E_\beta$ ssi $S(o_i, o_j) \geq \beta$.*

Definición 2.2 (Conjunto de vértices adyacentes a un vértice en G_β)**.** *Sean $G_\beta = \langle V, E_\beta \rangle$ un grafo de β-semejanza y $v \in V$, un vértice de G_β. El conjunto de vértices*

adyacentes de v, denotado por $v.Adj$, es el conjunto de vértices $u \in V$ tal que existe una arista $(v, u) \in E_\beta$.

Los vértices cuyo conjunto de vértices adyacentes sea nulo o vacío, se llaman vértices *aislados*. Adicionalmente, la cardinalidad del conjunto $v.Adj$ representa el *grado* de v.

Definición 2.3 (Sub-grafo)**.** *Sean $G_1 = \langle V_1, E_1 \rangle$ y $G_2 = \langle V_2, E_2 \rangle$ dos grafos. Se dice que G_1 es un* sub-grafo *de G_2 ssi se cumple que $V_1 \subseteq V_2$ y $E_1 \subseteq E_2$. En este caso se usa la notación $G_1 \subseteq G_2$.*

Definición 2.4 (Cubrimiento de G_β)**.** *Sean $G_\beta = \langle V, E_\beta \rangle$ un grafo de β-semejanza y $W = \{G_1, G_2, \ldots, G_k\}$ un conjunto de sub-grafos en el cual, $\forall i = 1..k$, $G_i \subseteq G_\beta$. El conjunto W es un cubrimiento de G_β ssi $\forall\ v \in V$, $\exists\ G_i = \langle V_i, E_i \rangle \in W$ tal que $v \in V_i$.*

Definición 2.5 (Componente β-conexa)**.** *Sean $G_\beta = \langle V, E_\beta \rangle$ un grafo de β--semejanza y $G' = \langle V', E' \rangle$ un sub-grafo en G_β. El sub-grafo G' es una componente β-conexa en G_β ssi satisface las siguientes condiciones:*

i) $\forall u, v \in V'$, $u \neq v$, existen $x_1, x_2, \ldots, x_q \in V'$, tal que $\forall i = 1..q-1$, $(x_i, x_{i+1}) \in E'$ y además $x_1 = u$ y $x_q = v$ o $x_1 = v$ y $x_q = u$.

ii) No existe otro sub-grafo de G_β, $G_1 = \langle V_1, E_1 \rangle$ con G_1 diferente de G', que satisfaga la condición i) y que además cumpla que $G' \subseteq G_1$.

Definición 2.6 (Componente β-conexa inducida por un vértice)**.** *Sean $G_\beta = \langle V, E_\beta \rangle$ un grafo de β-semejanza y v un vértice del mismo. La componente β-conexa inducida por el vértice v en G_β, es el sub-grafo $G' = \langle V', E' \rangle$ tal que G' es una componente β-conexa en G_β y además $v \in V'$.*

Definición 2.7 (Sub-grafo en forma de estrella)**.** *Sea $G_\beta = \langle V, E_\beta \rangle$ un grafo de β-semejanza y $G_1 = \langle V_1, E_1 \rangle$ un sub-grafo G_β. Se dice que G_1 es un sub-grafo en forma de estrella (s-grafo) ssi existe un vértice $v \in V_1$ tal que $\forall\ u \in V_1, u \neq v$ se cumple que $(v, u) \in E_1$. El vértice v es conocido como el* centro *del sub-grafo y el resto de los vértices son llamados* satélites*.*

Cuando un s-grafo solo esté compuesto por su centro, entonces se dirá que dicho sub-grafo es un s-grafo *degenerado*.

Definición 2.8 (Grafo de máxima β-semejanza)**.** *Un grafo de máxima β-semejanza, denotado por $G_{max-\beta} = \langle V, E_{max-\beta} \rangle$, es el grafo dirigido en el cual se cumple que $V = O$ y $\langle o_i, o_j \rangle \in E_{max-\beta}$ ssi $S(o_i, o_j) = max\{S(o_i, o_k) \mid o_k \in V \wedge o_k \neq o_i \wedge S(o_i, o_k) \geq \beta\}$.*

Definición 2.9 (Componente β-fuertemente conexa en $G_{max-\beta}$)**.** *Sean $G_{max-\beta} = \langle V, E_{max-\beta} \rangle$ un grafo de máxima β-semejanza y $G' = \langle V', E' \rangle$ un sub-grafo en $G_{max-\beta}$. El sub-grafo G' es una componente β-fuertemente conexa en $G_{max-\beta}$ ssi satisface las siguientes condiciones:*

i) $\forall\, v, u \in V'$, existen $p_1, p_2, \ldots, p_k$ vértices en V' tal que $v = p_1$, $u = p_k$ y existe una arista dirigida $\langle p_j, p_{j+1} \rangle \in E'$, $\forall\, j = 1..k-1$.

ii) No existe otro sub-grafo de $G_{max-\beta}$, $G_1 = \langle V_1, E_1 \rangle$ con G_1 diferente de G', que satisfaga (i) y que además cumpla que $G' \subseteq G_1$.

2.2. Algoritmos traslapados incrementales y dinámicos

En la literatura existen varios algoritmos de agrupamiento que han sido propuestos para la construcción de grupos traslapados (Aslam *et al.*, 1998; Zamir and Etziony, 1998; Pons-Porrata *et al.*, 2002; Gil-García *et al.*, 2003; Hammouda and Kamel, 2004; Pérez-Suárez and Medina-Pagola, 2007; Gago-Alonso *et al.*, 2007; Pérez-Suárez *et al.*, 2009; Gil-García and Pons-Porrata, 2010); sin embargo, la mayoría de ellos son estáticos y por tanto, no pueden procesar eficientemente adiciones y/o eliminaciones de objetos.

Los algoritmos de agrupamiento que permiten formar grupos traslapados y que son capaces de procesar cambios en la colección son: Star (Aslam *et al.*, 1998), STC (Zamir and Etziony, 1998), ISC (Pons-Porrata *et al.*, 2002), SHC (Hammouda and Kamel, 2004), ICSD (Pérez-Suárez *et al.*, 2009) y DHS (Gil-García and Pons-Porrata, 2010). 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, el resto son algoritmos no jerárquicos.

El algoritmo Star (Aslam *et al.*, 1998) es un algoritmo dinámico *basado en grafos* que ha sido utilizado para el filtrado (Aslam *et al.*, 2000) y la organización de información (Aslam *et al.*, 2004). Star representa la colección a través de su grafo de β-semejanza G_β (ver Definición 2.1) y obtiene un conjunto de grupos traslapados a través de la construcción de un cubrimiento de G_β (ver Definición 2.4). Para obtener este cubrimiento, Star utiliza sub-grafos en forma de estrella (ver Definición 2.7). En este contexto, cada s-grafo es interpretado como un grupo por el algoritmo Star.

Para formar este cubrimiento, Star construye una lista L que contiene todos los vértices del grafo. Posteriormente, Star selecciona iterativamente de L, utilizando una estrategia voraz, el vértice que forma el s-grafo más denso; *i.e.*, el sub-grafo que contiene el mayor número de satélites. Una vez que se selecciona un vértice, éste se adiciona a un conjunto R y es eliminado de la lista L, conjuntamente con sus vértices adyacentes. Este proceso termina cuando la lista L queda vacía. El s-grafo formado por cada vértice adicionado a R, determina un grupo en el agrupamiento final.

Sea $C = \{S_1, S_2, \ldots, S_k\}$ el cubrimiento actual de G. Cuando se adiciona o elimina un objeto de la colección, algunos s-grafos en C pudieran eliminarse, crearse o modificarse; por lo tanto, el cubrimiento actual necesita ser actualizado. Para actualizar el cubrimiento C, Star inicializa la lista L con todos los vértices que pudieran formar nuevos s-grafos. Cuando se elimina un objeto de la colección, L se inicializa con todos los satélites que fueron adyacentes al vértice eliminado. En otro caso, cuando se adiciona un objeto, la lista L se inicializa con el vértice adicionado y con los satélites adyacentes de éste. Una vez formada la lista L, Star extrae iterativamente de L el vértice v de mayor grado y comprueba las siguientes condiciones:

1) v no pertenece a ningún s-grafo contenido en C.

2) El s-grafo determinado por v es más denso que todos los s-grafos del cubrimiento actual, que contienen a v como satélite.

Si v satisface al menos una de las condiciones anteriores, entonces es seleccionado para cubrir a G_β; *i.e.*, el s-grafo determinado por v se adiciona al conjunto C. Adicionalmente, cuando v satisface (2), todos los s-grafos del cubrimiento actual, que contienen a v como satélite son eliminados de C; los

satélites de estos s-grafos eliminados son adicionados a L. El proceso de actualización del cubrimiento termina cuando la lista L queda vacía.

El algoritmo Star tiene varias limitaciones. La primera limitación es que obtiene grupos con alto traslape. La segunda limitación es que, cuando se adiciona o elimina más de un objeto a la vez, Star tiene que procesar los cambios uno por uno y actualizar el cubrimiento después de cada cambio; *i.e.*, Star no es capaz de procesar múltiples cambios. Es importante aclarar qué se entiende por múltiples cambios y por qué es importante ser capaz de procesarlos.

Supóngase que se desea adicionar un conjunto de objetos O a una colección ya agrupada. Para procesar estos cambios, Star adiciona los objetos de O uno a uno y actualiza el agrupamiento después de cada adición. Es importante notar que, si más de un cambio afectó a los mismos grupos entonces, dichos grupos se actualizarán más de una vez; por lo tanto, esta estrategia consumirá mucho tiempo de procesamiento. Una alternativa mejor sería permitir que el algoritmo adicione todos los objetos a la colección y posteriormente actualice solo una vez los grupos afectados. Es importante notar que la situación descrita anteriormente puede suceder también con eliminaciones o con una combinación de adiciones y eliminaciones de objetos.

Otra limitación del algoritmo Star es que construye muchos grupos, los cuales tienen un bajo promedio de elementos por grupo. Si se analiza la estrategia de cubrimiento de Star, cuando un vértice v es seleccionado para cubrir a G_β, tanto v como sus adyacentes son eliminados de L. Supóngase que los adyacentes de v tienen mayor grado que todos los vértices restantes en L. Como los vértices en L cubren menos vértices que los adyacentes eliminados, es muy posible que se seleccionen más s-grafos para cubrir a G_β, que los que se necesitaría seleccionar si los adyacentes de v no se eliminaran de L.

Otro algoritmo de agrupamiento traslapado, capaz de procesar cambios en la colección, es el algoritmo STC (Zamir and Etziony, 1998). Este algoritmo es incremental y fue desarrollado para agrupar colecciones de *snippets*. Los snippets son pequeños textos usados por sistemas de búsqueda como Google, para describir brevemente los resultados de las búsquedas.

Para agrupar una colección de snippets, el algoritmo STC utiliza una estrategia compuesta de tres pasos. En el primer paso, STC construye un *árbol*

de sufijos (Gusfield, 1997) que contiene a todos los sufijos de los snippets de la colección a agrupar. Un árbol de sufijos es un grafo conexo y sin ciclos (Aho *et al.*, 1983) que cumple las siguientes condiciones: (i) existe un nodo llamado raíz, (ii) cada nodo interno del árbol tiene al menos dos hijos, (iii) cada arista del árbol está etiquetada con una cadena de caracteres que es no vacía y (iv) cada nodo n_i contiene un conjunto de snippets de la colección y está etiquetado con una cadena de caracteres que es común a los snippets que el mismo contiene. Esta cadena de caracteres se obtiene al concatenar las etiquetas de las aristas que están en el camino del nodo raíz al nodo n_i.

En un segundo paso, STC construye, a partir del árbol de sufijos, el conjunto de todos los grupos *base*; los grupos base son los nodos que contienen dos o más snippets. Posteriormente, en el tercer paso STC realiza un proceso en el cual, utilizando una estrategia similar a la del algoritmo Single-link (Sibson, 1973), mezcla los grupos base que sean similares. Dos grupos base B_1 y B_2 son similares si el traslape de $B_1 \cup B_2$ con respecto a B_1 y a B_2, es mayor que β en ambos casos; β es un umbral de semejanza que se define a priori. El agrupamiento final está formado por los grupos que resultan de este proceso.

Cuando se adicionan snippets a la colección, STC actualiza el agrupamiento en cuatro pasos. En el primer paso actualiza el árbol de sufijos y determina el conjunto U de grupos base que fueron creados o modificados. Luego, en el segundo paso selecciona del resto de grupos base, aquellos k con mejor *ranking*. El ranking de un grupo base se determina a partir de la etiqueta del grupo base y de los snippets que el mismo contiene. Posteriormente, en el tercer paso se calcula la semejanza de los grupos base en U con los k grupos de mejor ranking. Por último, en el cuarto paso se reconstruye desde cero el agrupamiento final utilizando la estrategia de mezcla descrita anteriormente.

La principal limitación de STC está relacionada con la construcción del árbol de sufijos. Aunque la construcción de dicho árbol depende del conjunto de snippets a agrupar, por lo general resulta muy costosa cuando el número de snippets crece. Adicionalmente, es importante mencionar que, aunque STC es capaz de procesar múltiples adiciones, la estrategia de actualización del agrupamiento puede consumir mucho tiempo. Lo anterior se explica en el hecho de que, cada vez que se producen cambios en la colección, el conjunto final de grupos es reconstruido desde cero. Por último, STC nece-

sita ajustar los valores de β y k. Este ajuste depende de la colección que se está procesando y por lo tanto, puede resultar complejo.

El algoritmo compacto incremental (Pons-Porrata *et al.*, 2002) es otro algoritmo que permite obtener grupos con traslape y que es capaz de procesar adiciones a la colección de objetos. Este algoritmo, llamado ISC, representa la colección por su grafo de máxima β-semejanza $G_{max-\beta} = \langle V, E_{max-\beta} \rangle$ (ver Definición 2.8) y construye un cubrimiento de este grafo utilizando *conjuntos fuertemente compactos*. En este contexto, cada conjunto fuertemente compacto es un grupo.

Un conjunto $Q \subseteq V$ es un *conjunto fuertemente compacto* si satisface las siguientes condiciones:

1) $\forall\ v \in Q$, $\forall u \in V$ tal que $u \neq v$, si existe una arista $\langle v, u \rangle \in E_{max-\beta}$ entonces se cumple que $u \in Q$.

2) $\exists\ v \in Q$, tal que $\forall u \in Q$ existen vértices $x_1, x_2, \ldots, x_k \in V$ tales que $x_1 = v$, $x_k = u$ y $\langle x_q, x_{q+1} \rangle \in E_{max-\beta}$, $\forall q = 1..k-1$.

3) $\nexists\ Q' \subseteq V$ tal que Q' satisface las condiciones anteriores y $Q \subset Q'$.

Para construir el cubrimiento de $G_{max-\beta}$ el algoritmo ISC usa una estrategia compuesta por tres pasos. En el primer paso, ISC construye el conjunto $SC = \{C_1, C_2, \ldots, C_m\}$ de todas las componentes conexas en $G_{max-\beta}$ sin tener en cuenta la orientación de las aristas. Posteriormente, en el segundo paso se construye, para cada componente conexa C_k, el grafo dirigido $G_k = \langle V_k, E_k \rangle$, donde V_k es el conjunto de componentes fuertemente conexas de C_k (ver Definición 2.9) y $\forall V_{k_1}, V_{k_2} \in V_k$, existe una arista dirigida $\langle V_{k_1}, V_{k_2} \rangle \in E_k$ ssi existe un vértice $v \in V_{k_1}, u \in V_{k_2}$ tal que $\langle v, u \rangle \in E_{max-\beta}$. Por último, en el tercer paso se determinan las componentes conexas de $G_k = \langle V_k, E_k \rangle$; estas componentes constituyen los conjuntos fuertemente compactos que cubren a C_k. El agrupamiento final se forma con los conjuntos fuertemente compactos que cubren a cada componente $C_k \in SC$.

Sea F el conjunto de conjuntos fuertemente compactos que cubren a $G_{max-\beta}$. Cuando se adicionan nuevos objetos a la colección, pueden crearse nuevos conjuntos fuertemente compactos y otros existentes podrían desaparecer o modificarse; luego, el cubrimiento debe actualizarse. Para actualizar el agrupamiento ISC utiliza una estrategia compuesta por cuatro pasos.

En el primer paso, ISC construye el conjunto $U = \{C_1, C_2, \ldots, C_m\}$ de componentes conexas en $G_{max-\beta}$, sin tener en cuenta la orientación de las aristas, que contienen a los vértices adicionados. Sea $A = \{S_1, S_2, \ldots, S_t\}$ el conjunto de conjuntos fuertemente compactos de F, que contienen a uno o más vértices de alguna componente conexa en U. En el segundo paso se construye el conjunto de vértices $K = (\bigcup_{S_i \in A} S_i) \setminus (\bigcup_{C_i \in U} C_i)$ y se eliminan de F todos los conjuntos fuertemente compactos pertenecientes a A. Los vértices que pertenecen en K son los que formaran los nuevos grupos que se adicionarán al agrupamiento. Para formar estos grupos, en el tercer paso, ISC extrae del conjunto K las componentes conexas en $G_{max-\beta}$ sin tener en cuenta la orientación de las aristas y adiciona dichas componentes a U. Por último, en el cuarto paso se construyen, utilizando la estrategia anteriormente descrita, los conjuntos fuertemente compactos que cubren a cada componente en U. Estos nuevos conjuntos son los nuevos grupos que son adicionados a F; de esta forma, se obtiene el agrupamiento actualizado.

El algoritmo ISC tiene dos limitaciones. La primera limitación es que construye, al igual que el algoritmo Star, muchos grupos los cuales tienen un bajo promedio de elementos por grupo. Adicionalmente, ISC construye grupos que tienen alto traslape.

SHC (Hammouda and Kamel, 2004) es otro algoritmo de agrupamiento, capaz de formar grupos traslapados y de procesar cambios en la colección de objetos. SHC es un algoritmo incremental que se basa en el concepto de *histograma de semejanza de un grupo*.

El histograma de semejanza de un grupo c_i, denotado como H_{c_i}, es una representación estadística y concisa de la distribución de semejanzas existente entre los objetos de c_i. Este histograma está compuesto por un número de celdas que corresponden con intervalos fijos de semejanza. En cada celda se almacena el número de pares de objetos cuya semejanza está en el intervalo representado por dicha celda. Por ejemplo, supóngase que $c_i = \{a, b, c\}$ y que la semejanza entre los objetos es $s(a, b)$ =0.32, $s(a, c)$ =0.35 y $s(b, c)$ =0.53. Si las celdas de H_{c_i} corresponden con los intervalos $(0,0.10), (0.11,0.20), \ldots, (0.91,1.00)$ entonces, las celdas correspondientes a los intervalos $(0.31,0.40)$ y $(0.51,0.60)$ tendrán los valores 2 y 1 respectivamente.

Para agrupar una colección de objetos, el algoritmo SHC emplea una estrategia que utiliza el *cociente* del histograma de un grupo. Sea H_{c_i} el histo-

grama de un grupo c_i. El cociente de H_{c_i}, denotado por HR_{c_i}, es una medida de la cohesión del grupo c_i y se calcula de la siguiente forma:

$$HR_{c_i} = \frac{CS_\beta}{CS_{c_i}}, \tag{2.1}$$

donde CS_β es el número de pares de objetos de c_i cuya semejanza en H_{c_i} es mayor que cierto umbral β; CS_{c_i} es la suma de los valores de todas las celdas de H_{c_i}.

Cada vez que se adiciona un objeto O a la colección, SHC determina los grupos a los cuales se debería adicionar el objeto O. Para determinar dichos grupos, SHC simula la adición de O a cada grupo c_i y calcula el cociente de H_{c_i} antes (old-H_{c_i}) y después (new-H_{c_i}) de adicionar a O. Si new-H_{c_i} ≥old-H_{c_i} entonces el objeto O se adiciona a c_i. En otro caso, si old-H_{c_i}−new-$H_{c_i} \leq \epsilon$ y new-$H_{c_i} > \text{HR}_{min}$ entonces el objeto O se adiciona al grupo c_i. Si ninguna de las dos condiciones anteriores se cumple, entonces O no se adiciona a c_i. Una vez que se procesan todos los grupos, si el objeto O no se adicionó a ningún grupo entonces se forma un nuevo grupo que contiene solamente a O. En la explicación anterior, tanto ϵ y HR_{min} son parámetros del algoritmo.

El algoritmo SHC tiene varias limitaciones. La primera limitación es que SHC necesita ajustar los valores de β, ϵ y HR_{min}. Este ajuste depende de la colección que se está procesando y por lo tanto, puede resultar complejo. Otra limitación del algoritmo SHC es que construye grupos con alto traslape. Por último, al igual que el algoritmo Star, SHC no es capaz de procesar múltiples adiciones.

Otro algoritmo de agrupamiento que permite formar grupos con traslape y procesar adiciones a la colección de objetos, es ICSD (Pérez-Suárez *et al.*, 2009). Este algoritmo está basado en grafos y construye un agrupamiento a través del cubrimiento del grafo de β-semejanza G_β que representa a la colección de objetos. Para obtener este cubrimiento, ICSD utiliza s-grafos. Sin embargo, a diferencia del algoritmo Star, ICSD introduce un nuevo criterio para ordenar y seleccionar los s-grafos necesarios para el cubrimiento. Basado en este criterio, ICSD introduce además una nueva estrategia de cubrimiento de G_β.

Sea $G_\beta = \langle V, E_\beta \rangle$ un grafo de β-semejanza y v un vértice del mismo. El *strength* de v, denotado por $v.strength$, se calcula de la siguiente forma:

$$v.strength = \left| \{ u \in v.Adj \mid v.strength_{pre} \geq u.strength_{pre} \} \right|,$$

donde $v.strength_{pre}$ es el número de adyacentes de v que tienen un grado menor o igual que el de v; $u.strength_{pre}$ se define de la misma forma que $v.strength_{pre}$.

Para construir el cubrimiento de G_β, ICSD inicializa una lista L con todos los vértices de G_β que tienen un valor de strength mayor que cero. El cubrimiento de G_β se construye al procesar iterativamente la lista L en orden decreciente del valor de strength de los vértices. Sea X el conjunto de s-grafos que cubren a G_β; inicialmente $X = \emptyset$. En cada iteración se extrae de L el vértice v con mayor strength y se verifican en él las siguientes condiciones:

i) v no pertenece a ningún s-grafo contenido en X.

ii) Existe al menos un vértice $u \in v.Adj$, tal que u no pertenece a ningún s-grafo contenido en X.

Si v satisface alguna de las condiciones anteriores entonces se adiciona a X el s-grafo determinado por v. El proceso anterior termina cuando la lista L queda vacía. Una vez que se construye el conjunto X, éste se ordena ascendentemente respecto al número de satélites de cada s-grafo. Posteriormente, cada s-grafo de $S_i \in X$ es visitado, y si el conjunto $X \setminus S_i$ es un cubrimiento de G_β, entonces se elimina S_i de X. El agrupamiento final está constituido por los s-grafos que queden en X después de este proceso. Cuando se adicionan objetos a la colección, puede variar el valor de strength de los vértices, por lo que es necesario actualizar el cubrimiento actual de G_β. Para este propósito, ICSD construye el conjunto $U = \{C_1, C_2, \ldots, C_m\}$ de componentes conexas de G_β que contienen a los vértices adicionados y actualiza el cubrimiento de cada componente $C_k \in U$ utilizando la estrategia descrita anteriormente.

Las limitaciones del algoritmo ICSD son que construye un gran número de grupos y que dichos grupos tienen alto traslape.

El algoritmo DHS (Gil-García and Pons-Porrata, 2010) es el único algoritmo jerárquico que permite obtener jerarquías traslapadas y procesar cambios en la colección de objetos. DHS es un algoritmo dinámico y aglomerativo, diseñado para el agrupamiento de documentos y derivado de la metodología propuesta en (Gil-García *et al.*, 2005). Esta metodología propone la creación de un algoritmo jerárquico aglomerativo a partir de la aplicación, en

cada uno de los niveles de la jerarquía, de un algoritmo basado en grafos, el cual representa a los objetos utilizando el modelo VSM (Salton *et al.*, 1975).

Para construir la jerarquía de grupos, DHS considera que el nivel base de la jerarquía (nivel 1) es aquel en el que cada objeto de la colección forma un grupo independiente. A partir de este punto, cada nivel se construye utilizando los resultados del nivel anterior. En cada nivel $N_i > 1$, los objetos a agrupar son los grupos obtenidos en el nivel inmediato inferior. Para construir el agrupamiento en un nivel N_i cualquiera, se emplea una estrategia compuesta por tres pasos. En el primer paso se construye el gafo de de β-semejanza G_β que representa a la colección de objetos del nivel. Para determinar la semejanza entre dos grupos de objetos, DHS utiliza la medida group-average. Para calcular la semejanza entre dos objetos, DHS utiliza la medida del coseno.

Si G_β no tiene aristas, se detiene el proceso y la jerarquía queda formada por los niveles construidos anteriormente. En otro caso, si G_β es conexo entonces, en el segundo paso se construye el *grafo no dirigido de máxima β-semejanza* $\widehat{G}_{max-\beta} = \langle V, \widehat{E}_{max-\beta} \rangle$. Este grafo se obtiene a partir del grafo de máxima β-semejanza que representa a los objetos del nivel, pero considerando a las aristas como no orientadas. Una vez formado el grafo $\widehat{G}_{max-\beta}$, en el tercer paso se construye el cubrimiento de $\widehat{G}_{max-\beta}$ utilizando s-grafos.

Para obtener este cubrimiento DHS utiliza una estrategia similar a la del algoritmo Star (Aslam *et al.*, 1998). Sea X el cubrimiento del grafo $\widehat{G}_{max-\beta}$ que representa a los objetos del nivel actual; inicialmente $X = \emptyset$. La estrategia empleada por el algoritmo DHS para cubrir a $\widehat{G}_{max-\beta}$ parte de crear una lista L que contiene a todos los vértices de $\widehat{G}_{max-\beta}$. Posteriormente procesa esta lista iterativamente, extrayendo en cada iteración al vértice v de mayor grado. Sea $A = \{S_1, S_2, \ldots, S_t\}$ los s-grafos de X que contienen al vértice v. Si $A = \emptyset$ o el grado de v es mayor o igual que el de los centros de los s-grafos contenidos en A, entonces el s-grafo determinado por v se adiciona a X. Este proceso termina cuando la lista L queda vacía. El agrupamiento del nivel está constituido por cada s-grafo del conjunto X. Una vez obtenido el agrupamiento de un nivel, se pasa a construir el nivel siguiente.

Existen dos diferencias entre esta estrategia y la empleada por el algoritmo Star. La primera es que DHS permite que los centros de dos s--grafos sean adyacentes. La segunda es que Star realiza el cubrimiento sobre un grafo de β-semejanza y DHS sobre un grafo no dirigido de máxima

β-semejanza.

Cuando se adicionan o eliminan objetos a la colección, hay que actualizar el grafo G_β que representa a los objetos del primer nivel y, como consecuencia, actualizar el grafo $\widehat{G}_{max-\beta}$ del mismo nivel. Estas actualizaciones pueden provocar que se eliminen, adicionen o modifiquen los s-grafos que cubren a $\widehat{G}_{max-\beta}$; por lo tanto, hay que actualizar el cubrimiento del grafo $\widehat{G}_{max-\beta}$. La actualización del cubrimiento del nivel provoca la creación o eliminación de grupos y por tanto la adición o eliminación de objetos del nivel siguiente. Este proceso de actualización se repite hasta que el grafo G_β del nivel actual no tenga aristas. Es posible llegar a un nivel N_i donde el grafo G_β no tenga aristas y sin embargo no se haya alcanzado el tope de la jerarquía. En este caso, se eliminan todos los niveles mayores a N_i.

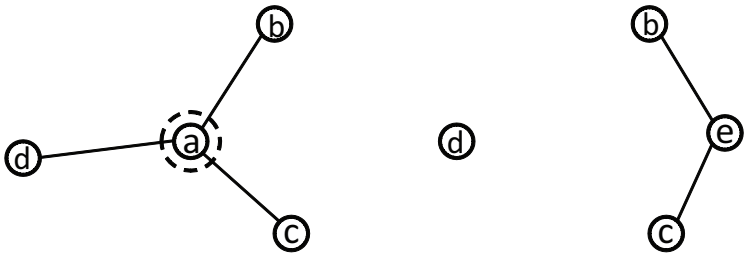
Para actualizar el cubrimiento del grafo $\widehat{G}_{max-\beta}$ que representa a un nivel, DHS utiliza una estrategia compuesta por dos pasos. Sea $F = \{S_1, S_2, \ldots, S_k\}$ el conjunto de s-grafos que cubre actualmente a $\widehat{G}_{max-\beta}$. Sea N y R los conjuntos de vértices adicionados y eliminados de $\widehat{G}_{max-\beta}$ respectivamente. Sea NE y RE los conjuntos de aristas adicionadas y eliminadas de $\widehat{G}_{max-\beta}$ respectivamente. En el primer paso, DHS construye una lista L que contiene a los vértices cuyos s-grafos pueden ser utilizados para actualizar F. Para formar la lista L se procesa cada s-grafo contenido en F. Sea $S_i \in F, i = 1..k$ un s-grafo que tiene como centro al vértice c. Para procesar a S_i se verifican las siguientes condiciones:

i) $\exists\, v \in V$, tal que existe una arista $(c, v) \in RE$.

ii) $\exists\, u, v \in V$, tal que existe una arista $(u, v) \in NE$ y $u \in c.Adj$.

Si S_i satisface algunas de las condiciones anteriores, entonces se elimina S_i de F y tanto c como sus satélites se adicionan a L. Adicionalmente, si S_i cumple (i) y $v \notin R$ entonces, se adiciona el vértice v a la lista L. Cuando se termina el procesamiento de los s-grafos de F entonces, se adicionan a L los vértices de N.

Una vez construida L, en el segundo paso DHS extrae iterativamente de L al vértice v de mayor grado. Sea $A = \{S_1, S_2, \ldots, S_t\}$ los s-grafos de F que contienen al vértice v. Si $A = \emptyset$ entonces, el s-grafo determinado por v se adiciona a X. Si $A \neq \emptyset$ pero el grado de v es mayor o igual que el de los



(a) Un grafo $\widehat{G}_{max-\beta}$

(b) Grafo $\widehat{G}_{max-\beta}$ después de los cambios

Figura 2.1: Ilustrando una limitación del algoritmo DHS.

centros de los s-grafos contenidos en A entonces, el s-grafo determinado por v se adiciona a X y todos los satélites de los s-grafos en A son adicionados a L. Este proceso termina cuando la lista L queda vacía.

DHS tiene varias limitaciones. Con el objetivo de acelerar el cálculo de la semejanza entre los grupos, DHS utiliza una propiedad demostrada en (Zhao and Karypis, 2002). Esta propiedad se puede aplicar solamente cuando los objetos se representan utilizando el modelo VSM (Salton *et al.*, 1975) y cuando se utiliza, como función de semejanza, a la medida del coseno (Berry, 2004). 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 de DHS es que puede dejar elementos sin cubrir en los niveles de la jerarquía. Esta limitación se explicará a través de un ejemplo. En la figura 2.1a se muestra el grafo no dirigido de máxima β-semejanza $\widehat{G}_{max-\beta}$ asociado a un nivel cualquiera de la jerarquía. En esta figura, el vértice centro está resaltado utilizando una línea discontinua. Supóngase que se elimina de este grafo al vértice a, que se adiciona el vértice e y que luego de actualizar $\widehat{G}_{max-\beta}$ se obtiene el grafo de la figura 2.1b.

Luego de procesar estos cambios, los conjuntos F, N, R, NE y RE son los siguientes: $F = \emptyset, N = \{e\}, NE = \{(b,e);(c,e)\}, R = \{a\}$ y $RE = \{(a,b);(a,c);(a,d)\}$. Siguiendo el procedimiento que utiliza DHS para formar la lista L, quedaría que $L = \{e\}$. Note que los vértices b, c y d no son adicionados a L, pues en el momento de procesar los conjuntos NE y RE, ya no existía el s-grafo que contenía a dichos vértices. Como el vértice e pertenece a L y es el de mayor grado, e resulta seleccionado y su s-grafo se adiciona

a F. Luego de esta operación, L queda vacía y se detiene el proceso de cubrimiento, quedando los vértices b, c y e cubiertos y el vértice d sin cubrir.

Por último, otra limitación del algoritmo DHS es que construye jerarquías con alto traslape, las cuales están formadas por muchos niveles y muchos grupos.

2.3. Síntesis y conclusiones

En este capítulo se presentaron los algoritmos relacionados con esta investigación y se describieron las limitaciones de cada uno. En la Tabla 2.1 se muestran las principales características de los algoritmos descritos en la sección anterior.

Tabla 2.1: Características de los algoritmos de agrupamiento que se relacionan con la investigación

Alg.	**Tipo de algoritmo**	**Número de parámetros**	**Complejidad computacional**	**Múltiples operaciones**	**Construye jerarquías**
Star	Dinámico	1	$O(n^2 \cdot \log^2 n)$	No	No
STC	Incremental	2	$O(n \cdot \log n)$	No	No
ISC	Incremental	1	$O(n^2)$	Si	No
SHC	Incremental	3	$O(n^2)$	No	No
ICSD	Incremental	1	$O(n^2)$	Si	No
DHS	Dinámico	1	$O(n^3)$	Si	Si

Es importante notar que, aunque existen varios algoritmos de agrupamiento que permiten formar grupos traslapados y que permiten procesan cambios en la colección, la mayoría de estos algoritmos solo permiten procesar adiciones. Los únicos algoritmos de agrupamiento que son dinámicos y que forman grupos traslapados son Star y DHS.

Como se pudo observar, las limitaciones de los algoritmos descritos están relacionadas con el gran número de grupos que los algoritmos construyen y con el alto traslape de dichos grupos. Por otro lado, hay algoritmos como el Star y el SHC que no pueden procesar múltiples cambios, lo cual dificulta su uso en aplicaciones que procesan colecciones que cambian con mucha frecuencia. Adicionalmente, hay algoritmos como el STC y el SHC

que necesitan ajustar valores para varios parámetros, lo cual hace difícil su aplicación en problemas prácticos.

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.

Con base en lo antes expuesto, se puede notar que es necesario desarrollar algoritmos de agrupamiento traslapado, jerárquicos y no jerárquicos, que no presenten las limitaciones de los algoritmos reportados en el estado del arte. En los siguientes capítulos se introducen los algoritmos que se han desarrollado como resultado de esta investigación. Primero, se introduce un algoritmo de agrupamiento que es dinámico y permite obtener grupos con traslape. Posteriormente, se introduce un algoritmo de agrupamiento jerárquico, que es dinámico y permite formar jerarquías traslapadas.

Capítulo 3

Algoritmos de agrupamiento con traslape

En este capítulo se introduce un nuevo algoritmo de agrupamiento (DClustR), que es dinámico y permite obtener grupos con traslape. La estructura de este capítulo es la que sigue: en la sección 3.1 se introducen los conceptos utilizados por DClustR. En la sección 3.2 se presentan, a través de un nuevo algoritmo estático llamado OClustR, las ideas utilizadas para la formación de grupos con traslape. En la sección 3.3 se introduce el algoritmo DClustR. En la sección 3.4, se describen los resultados experimentales y por último, en la sección 3.5, se presentan las conclusiones del capítulo.

3.1. Conceptos básicos

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 tal que $\forall o_i, o_j \in O, o_i \neq o_j$, $S(o_i, o_j)$=$S(o_j, o_i)$. Un *grafo de β-semejanza pesado* es un grafo no dirigido y pesado $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, w \rangle$ tal que:

i) $V = O$.

ii) $w : (V \times V) \longrightarrow [0, 1]$ es una función que asigna a cada arista no dirigida $(v, u) \in \widetilde{E}_\beta, v \neq u$, el valor de $S(v, u)$ como peso de la arista.

iii) $\forall o_i, o_j \in O, S(o_i, o_j) \geq \beta \Rightarrow (o_i, o_j) \in \widetilde{E}_\beta$.

Sea $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, w \rangle$ un grafo de β-semejanza pesado. Un *sub-grafo pesado en forma de estrella* (*ws-grafo*) en $\widetilde{G}_\beta$ es un sub-grafo $G^\star = \langle V^\star, E^\star, w \rangle$ que cumple las siguientes condiciones:

i) $V^\star \subseteq V$.

ii) $E^\star \subseteq \widetilde{E}_\beta$.

iii) $\exists\, c \in V^\star$ tal que $\forall v \in V^\star, v \neq c, (c, v) \in E^\star$.

El vértice c es llamado el *centro* del ws-grafo y los restantes vértices son llamados *satélites*. Los vértices aislados de $\widetilde{G}_\beta$ son considerados como ws-grafos *degenerados*.

Sean $v, u \in V$ dos vértices de $\widetilde{G}_\beta$ y sea $G^\star = \langle V^\star, E^\star, w \rangle$ el ws-grafo que tiene al vértice v como centro. Se dice que el vértice v *cubre* al vértice u ssi $u \in V^\star$.

Sea $G_v^\star = \langle V^\star, E^\star, w \rangle$ el ws-grafo que tiene al vértice v como centro. La semejanza intra-grupo de $G_v^\star$, denotada por $Intra\_sim(G_v^\star)$, es el peso promedio de todas las aristas que se pueden formar entre los vértices de $V^\star$ (Jo and Lee, 2007), es decir:

$$Intra\_sim(G_v^\star) = \frac{\sum_{z,u \in V^\star, z \neq u} w(z, u)}{\frac{|V^\star| \cdot (|V^\star| - 1)}{2}} \tag{3.1}$$

donde $w(z, u)$ es el peso de la arista no dirigida que conecta a los vértices z y u. Como el peso de una arista $(z, u) \notin E^\star$ es cero, la expresión (3.1) se puede reescribir como sigue:

$$Intra\_sim(G_v^\star) = \frac{\sum_{(z,u) \in E^\star} w(z, u)}{\frac{|V^\star| \cdot (|V^\star| - 1)}{2}} \tag{3.2}$$

Es importante notar que el denominador de la expresión (3.2) depende solamente del número de vértices contenidos en $G_v^\star$. Luego, mientras mayor

sea el número de aristas en $E^\star$, mayor será la semejanza intra-grupo de $G_v^\star$. Por lo tanto, el menor valor de la expresión (3.2) se alcanza cuando solamente existen aristas entre el centro v y los satélites de $G_v^\star$; en este caso, la semejanza intra-grupo de $G_v^\star$ puede ser calculada de la siguiente forma:

$$Intra\_sim(G_v^\star) = \frac{\sum_{u \in V^\star, u \neq v} w(v,u)}{\frac{|V^\star| \cdot (|V^\star| - 1)}{2}} \tag{3.3}$$

Si se analiza la expresión (3.3) puede notarse que, pequeños cambios en la cardinalidad del conjunto $V^\star$ pueden provocar grandes cambios en el valor de $Intra\_sim(G_v^\star)$. De acuerdo a la expresión (3.3), $Intra\_sim(G_v^\star)$ toma valores en el intervalo [0,1] que generalmente son muy cercanos a 0. Debido a esto, se decidió aproximar el valor de $Intra\_sim(G_v^\star)$ en la expresión (3.3) como el peso promedio de las aristas existentes entre el centro y cada satélite de $V^\star$, es decir:

$$Aprox\_Intra\_sim(G_v^\star) = \frac{\sum_{u \in V^\star, u \neq v} w(v,u)}{|V^\star| - 1} \tag{3.4}$$

Como puede observarse en la expresión (3.4), el denominador de la expresión (3.3) ha sido modificado de forma tal que, $Aprox\_Intra\_sim(G_v^\star)$ sigue tomando valores en el intervalo [0,1] pero, a diferencia de $Intra\_sim(G_v^\star)$, los valores de $Aprox\_Intra\_sim(G_v^\star)$ no decrecen tanto cuando aumenta el número de vértices en $V^\star$.

3.2. Algoritmo OClustR

En esta sección se introduce un nuevo algoritmo estático para el agrupamiento con traslape. Este algoritmo, llamado OClustR, representa la colección de objetos a través de un grafo de β-semejanza pesado $\widetilde{G}_\beta$ y construye un conjunto de grupos traslapado en dos pasos. En el primer paso, OClustR construye un conjunto inicial de grupos a través del cubrimiento de $\widetilde{G}_\beta$ utilizando ws-grafos. En el segundo paso, OClustR realiza un proceso para mejorar el conjunto inicial de grupos y así, formar el agrupamiento final. En este contexto, “mejorar el conjunto inicial de grupos” significa reducir el número de ws-grafos necesarios para cubrir a $\widetilde{G}_\beta$ y el traslape entre dichos sub-grafos.

A continuación se describe cómo OClustR obtiene el cubrimiento de $\widetilde{G}_\beta$ y posteriormente, cómo se mejora dicho cubrimiento.

3.2.1. Cubrimiento inicial del grafo

Una forma de construir un cubrimiento del grafo $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, w\rangle$ es encontrar el *conjunto dominante mínimo*. El conjunto dominante mínimo de $\widetilde{G}_\beta$ es el menor subconjunto de vértices $M \subseteq V$, tal que cualquier vértice de $\widetilde{G}_\beta$ pertenece a M o es adyacente al menos a un vértice que pertenece a M (Crescenzi and Kann, 1997). Sin embargo, obtener dicho conjunto es un problema NP-duro (Gil-García, 2005); por lo tanto, se aproximará dicho cubrimiento a través del cubrimiento de $\widetilde{G}_\beta$ usando ws-grafos.

Como se pudo observar en la sección 3.1, un ws-grafo está determinado por su vértice centro. Por lo tanto, el problema de encontrar un conjunto $W = \{G^\star_{c_1}, G^\star_{c_2}, \ldots, G^\star_{c_k}\}$ de ws-grafos, tal que W es un cubrimiento de $\widetilde{G}_\beta$, puede transformarse en el problema de construir un conjunto $C = \{c_1, c_2, \ldots, c_k\}$ tal que $c_i \in C$ es el centro de $G^\star_{c_i} \in W$, $\forall i = 1..k$. Dado que cada vértice de $\widetilde{G}_\beta$ puede formar un ws-grafo, todos los vértices en $V^\star$ deben ser procesados para construir el conjunto C. Para reducir el espacio de búsqueda y establecer un orden de selección entre los vértices de $V^\star$, OClustR introduce el concepto de *relevancia* de un vértice. Para definir la *relevancia* de un vértice v, se definirán primero los conceptos de *densidad relativa* y *compacidad relativa* de v

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

$$v.densityR = \frac{v.density}{|v.Adj|}, \tag{3.5}$$

donde $v.density$ denota a la densidad del vértice v y es el número de vértices $u \in v.Adj$ tales que $|v.Adj| \geq |u.Adj|$. La densidad de v expresa cuántos vértices adyacentes puede incluir el vértice v en el cubrimiento de $\widetilde{G}_\beta$. La densidad relativa de v toma valores en el intervalo [0,1] y determina qué fracción representa $v.density$ del número total de adyacentes que tiene v. Mientras más alto sea el valor de $v.densityR$, más adyacentes incluirá el vértice v en el cubrimiento de $\widetilde{G}_\beta$ y por lo tanto, mejor será el vértice v para el cubrimiento de $\widetilde{G}_\beta$. Dado que los vértices que no son contados en $v.density$ son cubiertos por otros vértices seleccionados previamente, un valor alto de $v.densityR$ in-

dica también que el ws-grafo que determina el vértice v tendrá bajo traslape con los ws-grafos seleccionados previamente.

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

$$v.compactnessR = \frac{v.compactness}{|v.Adj|}, \tag{3.6}$$

donde $v.compactness$ denota a la compacidad del vértice v y es el número de vértices $u \in v.Adj$ que forman un ws-grafo $G^{\star}_{u}$, tal que se cumple que $Aprox\_Intra\_sim(G^{\star}_{v}) \geq Aprox\_Intra\_sim(G^{\star}_{u})$; donde $G^{\star}_{v}$ es el ws-grafo determinado por el vértice v. La compacidad de v expresa, desde un punto de vista diferente a la densidad de v, cuántos vértices adyacentes puede incluir el vértice v en el cubrimiento de $\widetilde{G}_{\beta}$. La compacidad relativa de v toma valores en el intervalo [0,1] y determina la fracción que representa $v.compactness$, del total de vértices en $v.Adj$. Mientras más alto sea el valor de $v.compactnessR$, más adyacentes incluirá el vértice v en el cubrimiento de $\widetilde{G}_{\beta}$ y por lo tanto, mejor será el vértice v para el cubrimiento de $\widetilde{G}_{\beta}$. De forma similar a como ocurre con $v.densityR$, valores altos de $v.compactnessR$ indican que el ws-grafo generado por v comparte pocos vértices con los sub-grafos seleccionados previamente y por lo tanto, produce un bajo traslape con dichos sub-grafos.

El concepto de *relevancia* de un vértice usado por OClustR combina los criterios de densidad relativa y compacidad relativa del vértice. La *relevancia* de un vértice v, denotada por $v.relevance$, se calcula como:

$$v.relevance = \frac{v.densityR + v.compactnessR}{2}, \tag{3.7}$$

donde $v.densityR$ y $v.compactnessR$ son la densidad relativa y compacidad relativa del vértice v. Las propiedades $v.densityR$ y $v.compactnessR$ toman valores en [0,1] por lo tanto, la expresión (3.7) también toma valores en [0,1]. Valores altos de densidad relativa y de compacidad relativa corresponden con vértices que, de acuerdo a estos dos criterios, se deberían seleccionar para formar el cubrimiento de $\widetilde{G}_{\beta}$. A partir de la expresión (3.7) se puede inferir que, altos valores de relevancia corresponderán con vértices que tienen altos valores de densidad relativa y/o compacidad relativa; por lo tanto, los vértices con altos valores de relevancia serán más útiles para formar el cubrimiento de $\widetilde{G}_{\beta}$. Del análisis anterior se puede concluir que los vértices que tengan una relevancia cero tienen $v.densityR = 0$ y $v.compactnessR = 0$, por

lo tanto, estos vértices no se tienen que verificar. Otra conclusion importante es que los vértices deben de ser seleccionados en orden descendente de sus valores de relevancia.

A partir del análisis presentado hasta el momento, se introduce una estrategia que permite formar un cubrimiento de $\widetilde{G}_\beta$ en tres pasos. En el primer paso, se forma una lista L que contiene a los vértices no aislados que tienen un valor de relevancia mayor que cero; los vértices aislados constituyen ws--grafos degenerados y por lo tanto, son incluidos directamente en C. Luego, en el segundo paso se ordena la lista L descendentemente de acuerdo al valor de relevancia de los vértices. En el tercer paso, se visitan cada uno de los vértices de L. Durante este tercer paso, para cada vértice $v \in L$ se verifican las siguientes condiciones:

a) v no está cubierto todavía; *i.e.*, $v \notin C$ y $\nexists\, c \in C$, tal que $(c, v) \in \widetilde{E}_\beta$.

b) v está cubierto pero tiene al menos un vértice adyacente u que no está cubierto; *i.e.*, existe al menos un vértice $u \in v.Adj$ tal que: (i) $u \notin C$ y (ii) $\nexists\, c \in C$, tal que $(c, u) \in \widetilde{E}_\beta$. Esta condición evita la selección de vértices ya cubiertos que, en el momento de su análisis, tengan todos sus adyacentes cubiertos por vértices seleccionados previamente.

Si un vértice v cumple al menos una de las condiciones descritas anteriormente, entonces se adiciona el vértice v al conjunto C. El proceso anterior termina cuando todos los vértices de L fueron procesados.

3.2.2. Mejoramiento del cubrimiento obtenido

Una vez que se construye el cubrimiento de $\widetilde{G}_\beta$, se analiza el conjunto C con el objetivo de eliminar los vértices que son poco útiles para el cubrimiento. En este contexto, la utilidad de un ws-grafo se determinará en función del número total de satélites que conforman el ws-grafo y del número de satélites que están contenidos solamente en dicho ws-grafo. A continuación se describen los detalles que permiten definir concretamente cuándo un ws--grafo es poco útil y cómo proceder con su eliminación. Posteriormente, se presenta la estrategia que se propone para obtener el agrupamiento final a partir del cubrimiento inicial de G_β.

Al terminar el proceso de cubrimiento de $\widetilde{G}_\beta$, pueden existir en C algunos vértices que, en algún momento, fueron necesarios para completar el cubrimiento de $\widetilde{G}_\beta$, pero que ahora ya no lo son más. Si un vértice $c \in C$ está cubierto y forma un ws-grafo que tiene todos sus satélites cubiertos por otros vértices de C, entonces este ws-grafo es poco útil y se podría eliminar al vértice c del conjunto C y el grafo $\widetilde{G}_\beta$ permanecería cubierto.

Por otra parte, si $c \in C$ está cubierto y forma un ws-grafo que tiene "casi todos" sus satélites cubiertos por otros vértices en C, entonces aunque este ws-grafo es poco útil, c no puede ser eliminado de C. Note que, si c es eliminado de C entonces, los vértices $u \in c.Adj$ que solo están cubiertos por c se quedarían sin cubrir. No obstante, si estos vértices pudieran adicionarse de alguna forma al ws-grafo definido por otro vértice $c' \in C$, entonces el grafo $\widetilde{G}_\beta$ seguiría estando cubierto y por lo tanto, se podría eliminar a c del conjunto C. Para solucionar este caso primero se definirá cuándo un vértice $c \in C$ forma un ws-grafo que es poco util, es decir, que tiene "casi todos" sus satélites cubiertos por otros vértices de C y posteriormente, cuándo este tipo de vértice puede ser eliminado de C.

Definir cuándo un vértice $c \in C$ forma un ws-grafo que tiene "casi todos" sus satélites cubiertos por otros vértices de C no es trivial. Por lo general, lo que se entiende por "casi todos" puede cambiar en dependencia del problema concreto que se esté analizando e incluso, de los intereses del usuario. Sea c un vértice del conjunto C. Sea $c.Non\_shared$ el subconjunto de vértices de $c.Adj$ que solo son cubiertos por c y $c.Shared$ el subconjunto de vértices de $c.Adj$ que son cubiertos por al menos otro vértice de C. En esta investigación, se considerará que un vértice $c \in C$ forma un ws-grafo que tiene "casi todos" sus satélites cubiertos por otros vértices en C ssi $|c.Shared| > |c.Non\_shared|$. Cada vértice que satisfaga la condición anterior es llamado *prescindible*.

Para eliminar un vértice prescindible c de C tienen que cumplirse dos condiciones. La primera condición es que c tiene que estar cubierto por algún otro vértice de C. Note que si c no está cubierto, entonces tanto los vértices de $c.Non\_shared$ como el propio c quedarán descubiertos. La segunda condición es que los vértices de $c.Non\_shared$ tienen que poderse adicionar, de alguna forma, al ws-grafo determinado por al menos un vértice $c' \in C$. En lo siguiente se asumirá que cada vértice $c \in C$ tiene una lista de vértices $c.Linked$ y que el ws-grafo determinado por c contiene, además de los

vértices en $c.Adj$, a los vértices de $c.Linked$. La lista $c.Linked$ contiene los vértices que, sin ser adyacentes a c, están incluidos en el grupo que determina c. En lo siguiente estos vértices serán referidos como vértices *ligados* a c

Los ws-grafos candidatos para adicionar a los vértices de $c.Non\_shared$, son aquellos que están determinados por vértices $c' \in C$ que son adyacentes a c. Note que, de acuerdo a la primera condición, existe al menos un vértice $c' \in C$ que es adyacente a c. Si c tuviera más de un vértice $c' \in C$ adyacente, los vértices de $c.Non\_shared$ deberían ser adjuntados solamente a uno de ellos; en otro caso, aumentaría el traslape entre los ws-grafos resultantes. Cuando $|(c.Adj \cap C)| > 1$, se seleccionará para adjuntar los vértices no cubiertos, al vértice $c' \in (c.Adj \cap C)$ de mayor grado. Si más de un vértice $c' \in (c.Adj \cap C)$ tiene el mayor grado, cualquiera de ellos pudiera seleccionase; por simplicidad, en este trabajo se toma al primero de estos vértices. En lo siguiente, los vértices $c' \in (c.Adj \cap C)$ de mayor grado, serán conocidos como *conectores* de c.

Para eliminar de C los vértices prescindibles, se visitará cada vértice $c \in C$ y se eliminará de $c.Adj$ cada vértice prescindible u que tenga a c como conector, añadiendo los vertices de $u.Non\_shared$ a la lista $c.Linked$. Obviamente, esta estrategia será efectiva y consumirá menos tiempo, si los vértices de C son analizados en orden descendente de sus grados. Siguiendo este orden, se garantiza que el vértice c que se esté procesando sea un conector de todos los vértices prescindibles de $c.Adj$; en otro caso, dichos vértices prescindibles debieron haber sido eliminados en iteraciones previas. Cuando un vértice prescindible es eliminado de C, el número de vértices contenidos en $c'.Shared$, para cada vértice c' que quede en C, puede decrecer pero nunca aumentar. Por lo tanto, si mientras se analiza un vértice c se determina que un vértice $c' \in (c.Adj \cup C)$ no es prescindible entonces, c' no tiene que verificarse en próximas iteraciones pues no se volverá prescindible. Usando el razonamiento anterior se puede reducir el tiempo de procesamiento. Con base en el análisis realizado hasta este punto, se presenta a continuación la estrategia propuesta para mejorar el cubrimiento obtenido en la sección 3.2.1 y consecuentemente, obtener el agrupamiento final.

La estrategia que se propone está compuesta de dos pasos. En el primer paso, el conjunto C es ordenado descendentemente de acuerdo al grado de los vértices y cada vértice de C se marca como *no-analizado*. En el segun-

do paso, se procesa cada vértice $c \in C$ para eliminar de $c.Adj$ los vértices prescindibles. Para procesar al vértice c se visita cada vértice $u \in c.Adj$. Si u pertenece a C y está marcado como *no-analizado*, entonces se determina si u es prescindible o no. Si u es prescindible entonces, u es eliminado de C y los vértices en $u.Non\_shared$ son adicionados a la lista $c.Linked$. En otro caso, si u no es prescindible entonces, u se marca como *analizado*. Cuando todos los vértices en $c.Adj$ son analizados, el vértice c se marca como *centro*.

Finalmente, para construir el agrupamiento final se recorren los vértices de $\widetilde{G}_\beta$. Cada vértice v marcado como *centro*, junto con los vértices en $v.Adj$ y en $v.Linked$, determina un grupo en el agrupamiento final. El pseudocódigo del algoritmo OClustR se muestra en el Algoritmo 3.1.

Algoritmo 3.1: Algoritmo OClustR

Input: $O = \{o_1, o_2, \ldots, o_n\}$ - colección de objetos, β - umbral de semejanza
Output: SC - conjunto de grupos, $\widetilde{G}_\beta$ - grafo que representa a O

```
   // Construyendo la lista de candidatos
 1 "Construir G̃_β = ⟨V, Ẽ_β, w⟩ a partir de O y β";
 2 "Marcar los vértices en G̃_β como satélites";
 3 "Calcular la relevancia de cada vértice en G̃_β";
 4 SC := ∅;
 5 C := {v ∈ V | v.Adj = ∅};
 6 L := {v ∈ V | v.relevance > 0};
   // Construyendo el cubrimiento de G̃_β
 7 "Ordenar L en orden descendente de acuerdo a la relevancia";
 8 foreach vértice v ∈ L do
 9 |   if v satisface las condiciones a) o b) then C := C ∪ {v};
10 end
   // Eliminando los vértices prescindibles
11 "Ordenar C en orden descendente de acuerdo al grado";
12 "Marcar cada vértice en C como no-analizado";
13 foreach vértice c ∈ C do
14 |   c.Linked := ∅;
15 |   foreach vértice v ∈ c.Adj do
16 |   |   if v ∈ C y v está marcado como no-analizado then
17 |   |   |   if v es prescindible then
18 |   |   |   |   c.Linked := c.Linked ∪ v.Non_shared;
19 |   |   |   |   C := C \ {v};
20 |   |   |   end
21 |   |   |   else "Marcar v como analizado";
22 |   |   end
23 |   end
24 |   "Marcar c como centro";
25 end
   // Construyendo el agrupamiento final
26 foreach vértice v ∈ V do
27 |   if v está marcado como centro then SC := SC ∪ {{c} ∪ c.Adj ∪ c.Linked};
28 end
```

3.2.3. Análisis de la complejidad computacional

Para determinar la complejidad computacional del Algoritmo 3.1, se analizará cada uno de sus pasos. Sea n el número de objetos de la colección $O = \{o_1, o_2, \ldots, o_n\}$; *i.e.*, $|O| = n$. Para construir $\widetilde{G}_\beta$ en el paso 1, se debe calcular la semejanza entre todo par de objetos de O. Por lo tanto, el número de operaciones que se realiza en este paso es $T_1(n) = n^2$; así, $T_1(n)$ es $O(n^2)$. Durante este paso se puede calcular, usando la expresión (3.4), la semejanza intra-grupo aproximada del ws-grafo inducido por cada vértice de $\widetilde{G}_\beta$, sin que se afecte $T_1(n)$. Para marcar cada vértice de $\widetilde{G}_\beta$ como satélite se necesita realizar un número de operaciones $T_2(n) = n$; luego, $T_2(n)$ es $O(n)$.

Para calcular la relevancia de cada vértice $v \in V$, en el paso 3, 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 3 es en el peor caso, $T_3(n) = n^2$; luego, $T_3(n)$ es $O(n^2)$.

Los pasos 5 y 6 pueden realizarse en un mismo ciclo, verificando el grado y la relevancia de cada vértice en V. Por lo tanto, el número total de operaciones realizadas en estos pasos es $T_{5-6}(n) = n$; por consiguiente, $T_{5-6}(n)$ es $O(n)$. El paso 7 puede realizarse en $O(n \cdot \log n)$ utilizando el algoritmo mergesort (Knuth, 1973); por lo tanto, $T_7(n)$ es $O(n \cdot \log n)$.

En el proceso en el cual los candidatos de L son analizados (pasos 8--10), 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 $T_{8-10}(n) = |L| \cdot n$. Dado que, en el peor de los casos $|L| = n$, el número de operaciones realizadas en los pasos 8-10 es $T_{8-10}(n) = n^2$; luego, $T_{8-10}(n)$ es $O(n^2)$. Como el grado de un vértice es un número entero en el intervalo [0,$n-1$], si se sigue el principio del palomar (*pigeonhole principle*), enton-

ces el paso 11 puede realizarse con un número de operaciones $T_{11}(n) = n$; luego, $T_{11}(n)$ es $O(n)$. El número de operaciones realizadas en el paso 12 es $T_{12}(n) = |C|$. Si en el peor de los casos todos los vértices de L fueran seleccionados como centro entonces, $|C| = n$ y por consiguiente, $T_{12}(n) = n$; así, $T_{12}(n)$ es $O(n)$.

Para calcular el número de operaciones realizadas en los pasos 13-25, en lo siguiente nombrado como *ciclo-A*, se requiere un análisis más profundo. Sea $|C| = q_0$ el número de vértices que se seleccionan en los pasos 8--10. En la primera iteración de *ciclo-A*, durante el ciclo de los pasos 15-23, en lo siguiente nombrado como *ciclo-B*, cada vértice $v \in \{c.Adj \cap C\}$ que está marcado como *no-analizado* resulta eliminado de C o marcado como *analizado* cuando se procesa a v en los pasos 17-21. Por lo tanto, el conjunto de vértices $v \in \{c.Adj \cap C\}$ que son procesados para un vértice c en el *ciclo-B*, durante la primera iteración de *ciclo-A*, puede ser dividido en los conjuntos 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 fueron eliminados de C por ser prescindibles. Adicionalmente, luego de la ejecución de *ciclo-B*, en la primera iteración de *ciclo-A*, quedan en C 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, existen $|M_1|$ vértices en C marcados como *analizados*, $|Z_1|$ vértices en C marcados como *no-analizados* y $|R_1|$ vértices fueron eliminados de C por ser prescindibles. Es importante notar que $|M_1| + |R_1| + |Z_1| = q_0$. Posteriormente, como solo quedan $|Z_1|$ vértices marcados como *no-analizados*, cualquier vértice que se procese en los pasos 17-21 de ciclo-B en la segunda iteración de ciclo-A, pertenecerá al conjunto Z_1; esto significa que $|M_2|+|R_2|+|Z_2| = |Z_1|$. Siguiendo un razonamiento similar se obtiene que en la tercera iteración $|M_3| + |R_3| + |Z_3| = |Z_2|$. Generalizando, para todas las iteraciones $i > 1$ se cumple que $|M_i| + |R_i| + |Z_i| = |Z_{i-1}|$.

El número de operaciones realizadas en una iteración i de *ciclo-A* es $F_i(n) = A + B$, donde A es el número de operaciones realizadas en *ciclo-B* para los vértices en $c.Adj \cup C$ marcados como *no-analizados* y B es el resto de las operaciones realizadas en *ciclo-B*. Para determinar cuándo un vértice v es prescindible o no, se necesita visitar cada vértice en $v.Adj$. Dado que el número de vértices en $c.Adj \cup C$ que están marcados como *no-analizados* al comienzo de *ciclo-B* es $|M_i| + |R_i|$, se puede concluir que $A = n \cdot (|M_i| + |R_i|)$.

Como para los vértices que no satisfacen las condiciones del paso 16 no se realiza ninguna otra operación, se puede concluir que $B = n - (|M_i| + |R_i|)$.

Hasta el momento, se tiene que el número de operaciones realizadas en una iteración i de *ciclo-A* es $F_i(n) = n \cdot (|M_i| + |R_i|) + n - (|M_i| + |R_i|)$. 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 13-25 es:

$$T_{13-25}(n) = \sum_{i=1}^{q} F_i(n) = \sum_{i=1}^{q} (n \cdot (|M_i| + |R_i|) + n - (|M_i| + |R_i|))$$

$$T_{13-25}(n) = \sum_{i=1}^{q} (n \cdot (|M_i| + |R_i|)) + \sum_{i=1}^{q} n - \sum_{i=1}^{q} (|M_i| + |R_i|)$$

así,

$$T_{13-25}(n) = n \cdot \sum_{i=1}^{q} (|M_i| + |R_i|) + n \cdot q - \sum_{i=1}^{q} (|M_i| + |R_i|) \tag{3.8}$$

Como fue mencionado anteriormente, se cumple que:

$$\begin{array}{rcl} |M_1| + |R_1| + |Z_1| & = & q_0 \\ |M_2| + |R_2| + |Z_2| & = & |Z_1| \\ & \vdots & \\ \left|M_q\right| + \left|R_q\right| + \left|Z_q\right| & = & \left|Z_{q-1}\right| \end{array} \tag{3.9}$$

Sumando las ecuaciones en (3.9) se obtiene:

$$\sum_{i=1}^{q} (|M_i| + |R_i|) + \sum_{i=1}^{q} |Z_i| = q_0 + \sum_{i=1}^{q-1} |Z_i|$$

$$\sum_{i=1}^{q} (|M_i| + |R_i|) + \left|Z_q\right| = q_0$$

luego,

$$\sum_{i=1}^{q} (|M_i| + |R_i|) = q_0 - \left|Z_q\right| \tag{3.10}$$

Substituyendo (3.10) en la ecuación obtenida en (3.8) se tiene que:

$$T_{13-25}(n) = n \cdot (q_0 - \left|Z_q\right|) + n \cdot q - (q_0 - \left|Z_q\right|)$$

$$T_{13-25}(n) = n \cdot q_0 - n \cdot \left|Z_q\right| + n \cdot q - q_0 + \left|Z_q\right|$$

agrupando términos de igual signo:

$$T_{13-25}(n) = n \cdot q_0 + n \cdot q + \left|Z_q\right| - (n \cdot \left|Z_q\right| + q_0)$$

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

$$T_{13-25}(n) \leq n \cdot q_0 + n \cdot q + \left|Z_q\right| \leq n^2 + n^2 + n$$

por tanto:

$$T_{13-25}(n) \leq 2 \cdot n^2 + n \tag{3.11}$$

A partir de (3.11) se puede concluir que $T_{13-25}(n)$ es $O(n^2)$. Para construir el agrupamiento final en los pasos 26-28, se visitan los vértices en $v.Adj$ y en $v.Linked$, de cada vértice $v \in V$ que esté marcado como *centro*; para los vértices no marcados como *centro* no se realiza ninguna operación. Luego, el número de operaciones realizadas en los pasos 26-28 es:

$$T_{26-28}(n) = \sum_{i=1}^{k} |v_i.Adj| + |v_i.Linked| + n - k$$

donde k es el número de vértices marcados como centro en $\widetilde{G}_\beta$. Para cualquier vértice v marcado como centro, se cumple que $v.Adj \cap v.Linked = \emptyset$. Por lo tanto, se tiene que $|v_i.Adj| + |v_i.Linked| \leq n$. Del análisis anterior se tiene que:

$$T_{26-28}(n) \leq \sum_{i=1}^{k} n + n - k$$

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

$$T_{26-28}(n) \leq n^2 \tag{3.12}$$

A partir de (3.12) se puede concluir que T_{26-28} es $O(n^2)$. Finalmente, el número total de operaciones realizadas en Algoritmo 3.1 es $T_t(n) = T_1(n) + T_2(n) + T_3(n) + T_{5-6}(n) + T_7(n) + T_{8-10}(n) + T_{11}(n) + T_{12}(n) + T_{13-25}(n) + T_{26-28}(n)$. Por la regla de la suma se tiene que $T_t(n)$ es $O(T_1(n), T_2(n), T_3(n), T_{5-6}(n), T_7(n), T_{8-10}(n), T_{11}(n), T_{12}(n), T_{13-25}(n), T_{26-28}(n))$. Por lo tanto, $T_t(n)$ es $O(n^2)$ y la complejidad computacional del algoritmo OClustR es $O(n^2)$.

3.3. Algoritmo dinámico DClustR

En esta sección se introduce un nuevo algoritmo de agrupamiento, nombrado DClustR, que es dinámico y permite obtener grupos con traslape. DClustR construye un conjunto de grupos traslapados utilizando los conceptos introducidos para el algoritmo OClustR en la sección anterior. Adicionalmente, en DClustR se introduce una estrategia para la actualización de dichos grupos cuando ocurren múltiples adiciones y/o eliminaciones de objetos.

La estrategia para actualizar el agrupamiento, cuando ocurren cambios en la colección, consiste en detectar cuáles son las componentes β-conexas que contienen a los grupos afectados por los cambios y, posteriormente, actualizar el cubrimiento de dichas componentes. Para actualizar el cubrimiento de una componente β-conexa $G' = \langle V', E' \rangle$ se emplean cuatro pasos. En el primer paso se recorren los vertices de G' y se forma el conjunto de vértices $C' = \{v_1, v_2, \ldots, v_k\}$ que determinan el cubrimiento previo de G'; estos vertices están marcados como *centro*. Luego, en el segundo paso se construye una lista L' de vértices candidatos a ser adicionados a C'. En el tercer paso, se actualiza el conjunto C' eliminando algunos de sus vértices y adicionando algunos otros vértices de L'. Por último, en el cuarto paso se eliminan de C' los vértices prescindibles y con los vértices restantes, se construyen los grupos que cubren a la componente.

En la siguiente sub-sección, primero se analizan cuáles son las componentes β-conexas que deben de ser analizadas cuando ocurren cambios en la colección y, posteriormente, se describe la estrategia utilizada para actualizar el cubrimiento de cada una de estas componentes. Por último, se presenta el pseudocódigo del algoritmo DClustR.

3.3.1. Actualización del cubrimiento

Sea O una colección de objetos, agrupada utilizando la estrategia descrita en la sección anterior. Sea $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, w \rangle$ el grafo de β-semejanza pesado que representa a O y C el conjunto de vértices marcados como *centro* que determinan el cubrimiento actual de $\widetilde{G}_\beta$. Para actualizar el agrupamiento actual cuando se realizan cambios en la colección O, es importante conocer cómo dichos cambios podrían afectar el cubrimiento actual de $\widetilde{G}_\beta$. En el análisis

que se presenta en esta sección se asume que pueden realizarse al mismo tiempo varias adiciones y/o eliminaciones de objetos.

Las eliminaciones y adiciones de objetos modifican la topología del grafo $\widetilde{G}_\beta$. La adición de un objeto o_j a la colección implica la adición de un nuevo vértice v a $\widetilde{G}_\beta$, el cálculo de la semejanza entre v y el resto de los vértices de $\widetilde{G}_\beta$ y, consecuentemente, la posible adición de aristas relativas a v. Por otra parte, la eliminación de un objeto de la colección implica la eliminación del vértice que representa a dicho objeto y la eliminación de las aristas relativas a dicho vértice. Como el procesamiento de eliminaciones y adiciones conlleva a la eliminación o adición de aristas en $\widetilde{G}_\beta$, se puede concluir que, luego de dicho procesamiento, se deberá recalcular la relevancia de algunos vértices de $\widetilde{G}_\beta$.

Cuando algunos vértices son adicionados/eliminados de $\widetilde{G}_\beta$, puede ocurrir al menos una de las siguientes situaciones:

1) Algunos vértices quedan descubiertos; *i.e.*, no pertenecen a ningún grupo formado por los vértices en C. Esta situación pasa en dos casos: (i) existe al menos un vértice adicionado que no es adyacente a ningún vértice en C o (ii) todos los vértices de C que cubrían a algún vértice especifico son eliminados.

2) La relevancia de algunos vértices cambia y como resultado, aparece al menos un vértice $v \notin C$ con una relevancia mayor que: (i) al menos un vértice en $v.Adj \cap C$ o (ii) al menos un vértice $u \in C$ que cubre vértices en $v.Adj$.

Para actualizar el agrupamiento cuando ocurre la situación 1), se debe seleccionar para cada vértice no cubierto v, algún vértice del conjunto $v.Adj \cup \{v\}$ y consecuentemente, adicionar el vértice seleccionado a C. Por otra parte, para saber cómo actualizar el agrupamiento cuando ocurre la situación 2), se requiere un análisis más profundo.

Para que un vértice v cambie su relevancia, primero debe cambiar su densidad o su compacidad. Un vértice v puede cambiar su densidad si cambia el valor de $v.density$ o $|v.Adj|$. Análogamente, un vértice v puede cambiar su compacidad si los valores de $v.compactness$ o $|v.Adj|$ cambian. Un cambio en el valor de $|v.Adj|$ solo se logra si se adicionan o eliminan vértices adya-

centes a v. Por otra parte, los valores de $v.density$ y $v.compactness$ pueden cambiar si algunas de las siguientes condiciones ocurren:

i) Se adicionan o eliminan vértices en el conjunto $v.Adj$.

ii) Existe al menos un vértice $u \in v.Adj$ al cual se le adicionó/eliminó al menos un vértice adyacente.

Como resultado del análisis anterior, se puede afirmar que los vértices que pueden cambiar su valor de relevancia pertenecen a las componentes β-conexas que contienen a: (i) los vértices adicionados a $\widetilde{G}_\beta$ o (ii) los vértices que eran adyacentes a los vértices eliminados de $\widetilde{G}_\beta$. Por lo tanto, para actualizar el cubrimiento de $\widetilde{G}_\beta$ solo se tiene que actualizar el cubrimiento de las componentes β-conexas que contienen a los vértices que pueden cambiar su relevancia. Es importante mencionar que algunos vértices incluidos en C pudieran convertirse en prescindibles cuando la colección cambia; por lo tanto, cuando se actualiza el cubrimiento de una componente β-conexa $G' = \langle V', E' \rangle$, se tiene que vaciar la lista $c'.Linked$ de cada vértice $c' \in (C \cap V')$. De esta forma, todos los vértices de G' que estaban en estas listas, quedarán descubiertos y por tanto, se necesitará adicionar nuevos vértices a C para cubrirlos.

Sea $G' = \langle V', E' \rangle$ una componente β-conexa cuyo cubrimiento debe ser actualizado, $C' = V' \cap C$ el conjunto de vértices cuyos ws-grafos cubren a los vértices de G' y $V'_s \subseteq (V' \setminus C')$ el conjunto de vértices de G' que tienen una relevancia no nula y que no pertenecen a C'. La estrategia utilizada para actualizar el cubrimiento de G' está compuesta de cinco pasos. En el primer paso, como se mencionó anteriormente, se recalcula la relevancia de los vértices de G' y se vacía la lista $c.Linked$ de cada vértice $c \in C'$. En el segundo paso, se construye la lista L' de vértices candidatos a ser incluidos en C'. Para formar esta lista, se procesan los conjuntos C' y V'_s como se explica a continuación.

En cada vértice $v \in V'_s$ se verifican las siguientes condiciones:

i) v no está cubierto.

ii) v tiene al menos un vértice adyacente que no está cubierto.

iii) v tiene al menos un vértice adyacente u tal que, el vértice $w \in C'$ que cubre a u y que tiene la mayor relevancia entre todos los vértices que cubren a u satisface que $v.relevance > w.relevance$.

Si v satisface algunas de las condiciones anteriores entonces, v es considerado como candidato y adicionado a L'. Todos los vértices $u \in v.Adj$ que hagan que v cumpla la condición *iii*) son marcados como *activados*. Los vértices marcados como *activados* son usados durante el análisis del conjunto C'.

Para analizar los vértices $c \in C'$, los vértices en $c.Adj$ son visitados. Cuando se visita un vértice $v \in c.Adj$, si v cumple que $v.relevance > c.relevance$ entonces, v se adiciona a L' y el vértice c es marcado como *débil*. Si cuando todos los vértices adyacentes de c fueron visitados, se tiene que c está marcado como *débil* o que c tiene al menos un adyacente marcado como *activado* entonces, c es eliminado de C'. Note que, el hecho de que c esté marcado como *débil* quiere decir que c pudiera ser cubierto por un vértice de mayor relevancia que el mismo c. De forma similar, el hecho de que c tenga algún vértice adyacente marcado como activado significa que dicho vértice pudiera ser cubierto por algún otro vértice de mayor relevancia que c. Por último, si c resultó eliminado de C y $c.relevance > 0$ entonces, c es considerado como candidato y es adicionado a L'; de esta forma, se garantiza que c pueda ser elegible para ser adicionado a C' y por lo tanto, ningún vértice de $c.Adj$ quede sin cubrirse.

Después de construir la lista L', en el tercer paso, se seleccionan los nuevos vértices necesarios para completar el cubrimiento de G'. Para seleccionar dichos vértices se utiliza la estrategia de selección de vértices del algoritmo OClustR (pasos 7-10 del Algoritmo 3.1). Los ws-grafos determinados por los nuevos vértices, junto con los ws-grafos determinados por los vértices que ya pertenecían a C', constituyen el cubrimiento actual de la componente β-conexa G'. Posteriormente, en el cuarto paso se mejora este cubrimiento mediante la eliminación de los vértices prescindibles de C'. Para eliminar los vértices prescindibles se utiliza la estrategia utilizada por OClustR para tal propósito (pasos 11-25 del Algoritmo 3.1). Sea C'' el conjunto de vértices que determinan los grupos que cubren a G'. En el quinto paso, los vértices de C'' son marcados como *centros* y los vértices en $V' \setminus C''$ son marcados como *satélite*. Finalmente, en el sexto paso los vértices de G' son marcados como *procesados*; de esta forma, se garantiza que una misma componente

no sea procesada más de una vez.

Cuando todas las componentes β-conexas afectadas por los cambios fueron procesadas, se visitan los vértices de $\widetilde{G}_\beta$ y los vértices que están marcados como centro determinan el agrupamiento actualizado. El pseudocódigo del algoritmo DClustR se muestra en el Algoritmo 3.2. Para procesar las adiciones y/o eliminaciones de objetos a la colección, se utiliza el procedimiento *UpdGraph*. Para actualizar el cubrimiento de las componentes β-conexas afectadas, se utiliza el procedimiento *UpdCovCompt*. Los pseudocódigos de los procedimientos *UpdGraph* y *UpdCovCompt* se muestran en los Algoritmos 3.3 y 3.4 respectivamente.

Algoritmo 3.2: Algoritmo DClustR

Input: $O = \{o_1, o_2, \ldots, o_n\}$ - colección de objetos, $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, w\rangle$ - grafo que representa a O, β - umbral de semejanza, R^+ - conjunto de objetos adicionados, R^- - conjunto de objetos eliminados

Output: O - colección de objetos actualizada, $\widetilde{G}_\beta$ - grafo actualizado que representa a O, SC - conjunto de grupos

```
   // Procesando las adiciones y/o las eliminaciones
 1 UpdGraph(O,G̃_β,β,R⁺,R⁻,M);
 2 "Marcar vértices en G̃_β como no-procesado";
   // Procesando componentes afectadas por los cambios
 3 foreach vértice v ∈ M do
 4     if v no está marcado como procesado then
 5         "Construir la componente β-conexa G' = ⟨V', E'⟩ inducida por v";
 6         if |V'| = 1 then "Marcar v como centro";
 7         else
 8             "Recalcular la relevancia de cada vértice de G'";
 9             "Formar V'_s y C' y vaciar c.Linked para cada vértice c ∈ C'";
10             "Construir lista L'";
11             C'' := ∅;
12             UpdCovCompt(G',L',C',C'');
13             "Marcar vértices en C'' como centro";
14             "Marcar vértices en V' \ C'' como satélite";
15         end
16         "Marcar vértices en G' como procesado";
17     end
18 end
   // Devolviendo el agrupamiento actualizado
19 SC := ∅;
20 foreach vértice v ∈ V do
21     if v está marcado como centro then SC := SC ∪ {{v} ∪ v.Adj ∪ v.Linked};
22 end
```

Como puede notarse en el Algoritmo 3.2, DClustR asume que existe un grafo de β-semejanza pesado $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, w\rangle$ que representa a la colección actual. Si no existiera una colección previa y es la primera vez que se va a procesar la colección O, entonces $\widetilde{G}_\beta$ es un grafo vacío; por lo que DClustR

Algoritmo 3.3: Procedimiento UpdGraph

Input: $O = \{o_1, o_2, \ldots, o_n\}$ - colección de objetos, $\widetilde{G}_\beta = \langle V, \widetilde{E}_\beta, w\rangle$ - grafo que representa a O, β - umbral de semejanza, R^+ - conjunto de objetos adicionados, R^- - conjunto de objetos eliminados

Output: O - colección de objetos actualizada, $\widetilde{G}_\beta$ - grafo actualizado que representa a O, M - conjunto de vértices

```
1  M := ∅;
   // Procesando las eliminaciones de objetos
2  foreach objeto u ∈ R⁻ do
3  |   "Eliminar de G̃_β al vértice v que representa al objeto u";
4  |   M := M ∪ {x | x ∈ v.Adj ∧ x no representa a ningún objeto de R⁻};
5  end
6  O := O \ R⁻;
   // Procesando las adiciones de objetos
7  foreach objeto u ∈ R⁺ do
8  |   "Crear vértice v en G̃_β para representar al objeto u";
9  |   "Calcular semejanza de v con el resto de los vértices de G̃_β";
10 |   M := M ∪ {v};
11 end
12 O := O ∪ R⁺;
```

Algoritmo 3.4: Procedimiento UpdCovCompt

Input: $G' = \langle V', E'\rangle$ - componente β-conexa, L' - lista de candidatos, C' - lista de vértices que cubren a G'

Output: C'' - lista actualizada de los vértices que cubren a G'

```
   // Seleccionando nuevos vértices para completar cubrimiento de G'
1  "Ordenar L' en orden descendente de acuerdo a la relevancia";
2  foreach vértice v ∈ L' do
3  |   if v satisface las condiciones a) o b) then C' := C' ∪ {v};
4  end
   // Eliminando los vértices prescindibles
5  "Ordenar C' en orden descendente de acuerdo al grado";
6  "Marcar cada vértice de C' como no-analizado";
7  foreach vértice c ∈ C' do
8  |   foreach vértice v ∈ c.Adj do
9  |   |   if v ∈ C' y v está marcado como no-analizado then
10 |   |   |   if v es prescindible then
11 |   |   |   |   c.Linked := c.Linked ∪ v.Non_shared;
12 |   |   |   |   C' := C' \ {v};
13 |   |   |   end
14 |   |   |   else "Marcar v como analizado";
15 |   |   end
16 |   end
17 |   C'' := C'' ∪ {c};
18 end
```

puede procesar una colección de objetos sin que exista un agrupamiento previo.

Note que DClustR procesa primero todas los cambios que ocurren en $\widetilde{G}_\beta$ producto de las adiciones y/o eliminaciones y, posteriormente, actualiza el cubrimiento de $\widetilde{G}_\beta$. Así, si más de un cambio afecta a un conjunto de grupos,

entonces dichos grupos se actualizarían solo cuando todos los cambios que los afectan han sido procesados y no cada vez que se procesa uno de estos cambios. De esta forma, DClustR reduce tiempo de actualización cuando ocurren múltiples adiciones y/o eliminaciones de objetos.

3.3.2. Análisis de la complejidad computacional

Para determinar la complejidad computacional del algoritmo DClustR, se analizarán cada uno de sus pasos. Sea $n^+ = |R^+|$ y $n^- = |R^-|$ el número de objetos adicionados y eliminados respectivamente. Sea n_0 el número de vértices de $\widetilde{G}_\beta$ antes de procesar las adiciones y/o eliminaciones. Sea $n = n_0 - n^- + n^+$, el número de vértices de $\widetilde{G}_\beta$ luego de procesar las adiciones y/o eliminaciones.

En el paso 1 se procesan, utilizando el procedimiento *UpdGraph*, las adiciones y/o eliminaciones de objetos. Como se puede observar en el Algoritmo 3.3, 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 R^+ y objetos en R^-. El número de operaciones realizadas para procesar las eliminaciones (pasos 1-5 del Algoritmo 3.3) es $F_1(n) = n^- \cdot (2 \cdot n_0) = 2 \cdot n^- \cdot n_0$. En el peor de los casos, si todos los objetos de $\widetilde{G}_\beta$ fueran eliminados, entonces $n^- = n_0$ y por tanto, el número de operaciones realizadas quedaría $F_1(n) = 2 \cdot \sum_{i=1}^{n} i = 2 \cdot \frac{n_0 \cdot (n_0 - 1)}{2} = n_0 \cdot (n_0 - 1)$; así, como $F_1(n) \leq n_0^2$ y $n_0^2 \leq n^2$, entonces $F_1(n)$ es $O(n^2)$.

Cuando se procesan las adiciones, es necesario calcular la semejanza entre cada vértice adicionado y el resto de vértices de $\widetilde{G}_\beta$. Como $n^+ \leq n$, el número de operaciones realizadas en los pasos 6-11 del Algoritmo 3.3 en el peor caso es $F_2(n) = n^2$; 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 utilizando la expresión (3.4), el valor de la semejanza intra-grupo aproximada del ws-grafo inducido por cada vértice de $\widetilde{G}_\beta$, sin afectar a $F_1(n)$ ni a $F_2(n)$.

Con base en el análisis anterior, se puede concluir que el número de operaciones realizadas por el Algoritmo 3.3 es $F_t(n) = F_1(n) + F_2(n)$. Por la regla se la suma se tiene que $F_t(n)$ es $O(F_1(n), F_2(n))$ y por tanto, $F_t(n)$ es $O(n^2)$. Luego, en el peor de los casos, el número de operaciones realizadas

en el paso 1 del algoritmo DClustR (algoritmo 3.2) es $T_1(n) = n^2$; por lo tanto, $T_1(n)$ 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 3-18. Sea $Z = \{G'_1, G'_2, \ldots, G'_k\}$ el conjunto donde $G'_i = \langle V'_i, E'_i \rangle, i = 1..k$ es la componente β-conexa inducida por el vértice $v_i \in M$ y $n_i = \left|V'_i\right|$ el número de vértices de la componente G'_i; note que $\sum_{i=1}^{k} n_i \leq n$. El número de operaciones realizadas en los pasos 3-18 depende de las operaciones realizadas en los pasos 5-16. El número de operaciones necesarias para formar la componente β-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 6-15 depende, en el peor de los casos, del número de operaciones realizadas en los pasos 8-14. Con base en el análisis realizado para calcular la complejidad computacional del algoritmo OClustR (ver sección 3.2.3), se puede afirmar que el número de operaciones necesarias para recalcular la relevancia de cada vértice $v \in V'$ en el paso 8 es, en el peor de los casos, $T_8(n_i) = n_i^2$; luego, $T_8(n_i)$ es $O(n_i^2)$.

Para formar los conjuntos V'_s y C' hay que visitar los vértices de G'_i y ver cuáles están marcados como *centro* 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 $c \in C'$ 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 $c \in C'$. Por tanto, el número de operaciones que se realizan en el paso 10 es $T_{10}(n_i) = n_i \cdot (\left|V'_s\right| + |C'|)$. Dado que $\left|V'_s\right| + |C'| \leq n_i$ entonces, en el peor de los casos $T_{10}(n_i) = n_i^2$; por tanto, $T_{10}(n_i)$ es $O(n_i^2)$.

Con base en el análisis realizado para calcular la complejidad computacional del algoritmo OClustR (ver sección 3.2.3), se puede afirmar que el número de operaciones realizadas en el paso 12 es $T_{12}(n_i) = n_i^2$; por tanto, $T_{12}(n_i)$ es $O(n_i^2)$. El número de operaciones realizadas en los pasos 13 y 14 es, en el peor de los casos, $T_{13}(n_i) = n_i$ y $T_{14}(n_i) = n_i$; luego, $T_{13}(n_i)$ es $O(n_i)$ y $T_{14}(n_i)$ es $O(n_i)$. En el paso 16 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 $T_{16}(n_i) = n_i$; así, $T_{16}(n_i)$ es $O(n_i)$.

El número de operaciones que se realizan en los pasos 5-16, para ac-

tualizar el cubrimiento de la componente β-conexa inducida por el vértice v_i, es:

$$T_{5-16}(n_i) = T_5(n_i) + T_8(n_i) + T_9(n_i) + T_{10}(n_i) + T_{12}(n_i) + T_{13}(n_i) + T_{14}(n_i) + T_{16}(n_i)$$

$$T_{5-16}(n_i) = n_i^2 + n_i^2 + n_i + n_i^2 + n_i^2 + n_i + n_i + n_i = 4 \cdot n_i^2 + 4 \cdot n_i$$

$$T_{5-16}(n_i) = 4 \cdot (n_i^2 + n_i)$$

luego, el número de operaciones realizadas para actualizar el cubrimiento de todas las componentes afectadas por los cambios (pasos 3-18) es:

$$T_{3-18}(n) = \sum_{i=1}^{k} T_{5-16}(n_i) = \sum_{i=1}^{k} 4 \cdot \left(n_i^2 + n_i\right) = 4 \cdot \sum_{i=1}^{k} \left(n_i^2 + n_i\right)$$

$$T_{3-18}(n) = 4 \cdot \sum_{i=1}^{k} n_i^2 + 4 \cdot \sum_{i=1}^{k} n_i$$

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:

$$T_{3-18}(n) \leq 4 \cdot \sum_{i=1}^{k} n_i^2 + 4 \cdot n \leq 4 \cdot \left(\sum_{i=1}^{k} n_i\right)^2 + 4 \cdot n$$

luego:

$$T_{3-18}(n) \leq 4 \cdot n^2 + 4 \cdot n \tag{3.13}$$

A partir de la expresión (3.13), se puede concluir que $T_{3-18}(n)$ es $O(n^2)$. Durante el análisis de la complejidad computacional del algoritmo OClustR, se demostró que el número de operaciones realizadas para formar los grupos es $O(n^2)$ (ver sección 3.2.3); luego, el número de operaciones realizadas en los pasos 20-22 es $T_{20-22}(n)$ es $O(n^2)$.

Hasta el momento, se tiene que el número total de operaciones realizadas en el algoritmo DClustR es $T_{1-22}(n) = T_1(n) + T_2(n) + T_{3-18}(n) + T_{20-22}(n)$. Aplicando la regla de la suma, se tiene que $T_{1-22}(n)$ es $O(T_1(n), T_2(n), T_{3-18}(n), T_{20-22}(n))$; por lo tanto, $T_{1-22}(n)$ 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 de agrupamiento no jerárquico, del estado del arte, que permiten formar grupos con traslape. Los algoritmos utilizados fueron: Star (Aslam *et al.*, 1998), ISC (Pons-Porrata *et al.*, 2002), SHC (Hammouda and Kamel, 2004), Estar (Gil-García *et al.*, 2003), Gstar (Pérez-Suárez and Medina-Pagola, 2007), ACONS (Gago-Alonso *et al.*, 2007) y ICSD (Pérez-Suárez *et al.*, 2009). En esta lista, además de los algoritmos descritos en el capítulo 2, se incluyeron los algoritmos Estar, Gstar y ACONS; estos son algoritmos estáticos que han obtenidos buenos resultados en la construcción de grupos con traslape. Es importante mencionar que, dado que DClustR utiliza la misma estrategia que OClustR para construir los grupos, estos experimentos también muestran el comportamiento del algoritmo OClustR de acuerdo a los tres criterios utilizados.

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 y ICSD; 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. El algoritmo Star es el único algoritmo dinámico, reportado en la literatura, que permite 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 que se propongan deben ser evaluados sobre colecciones en las que pueda existir traslape de forma natural. Por lo anterior, los algoritmos propuestos en este capitulo se evaluarán 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. El corpus AFP fue descargado del sitio http://trec.nist.gov y la misma 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 el misma 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.

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” y que tenían asociado al menos un tópico en el *ground-truth*, (3) la colección Reu-Tr se construyó utilizando las noticias del corpus Reuters que tenían la etiqueta “Train” y que tenían asociado al menos un tópico en el *ground-truth*, (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 que tenian asociado al menos un tópico en el *ground-truth*, (6) la colección cacm se construyó utilizando los documentos del corpus CACM que tenían asociado al menos un tópico en el *ground-truth*, (7) la colección TDT se construyó usando todas las noticias del corpus TDT2 que tenían asociado al menos un tópico en el *ground-truth*. Por último, se construyeron a partir de la colección TDT cin-

co 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 3.1. 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 cada colección. La columna etiquetada como "Traslape" representa el promedio de clases en las que está incluido cada documento de la colección (Gil-García and Pons-Porrata, 2010). El *ground-truth* de cada colección se distribuye junto con la colección.

Tabla 3.1: Características de las colecciones de prueba que se utilizarán en los experimentos

Colección	**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) (Salton *et al.*, 1975). Los términos de los documentos representan los lexemas de las palabras que ocurren al menos una vez en toda la colección. Estos lexemas fueron extraídos de los documentos utilizando 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 (Greengrass, 2001). Para determinar la semejanza entre dos documentos se utilizó la medida del coseno (Berry, 2004).

3.4.2. Evaluación de algoritmos de agrupamiento

Existen tres tipos de medidas de calidad reportadas en la literatura: *externas*, *internas* y *relativas* (Pfitzner *et al.*, 2009).

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. Algunos ejemplos de este tipo de medidas son Purity (Zhao and Karypis, 2001), Jaccard coefficient (Halkidi *et al.*, 2001) y F1-measure (Larsen and Aone, 1999), entre otras.

Las medidas relativas no dependen de información externa para evaluar los algoritmos de agrupamiento. Estas medidas evalúan un algoritmo mediante la comparación de los grupos que dicho algoritmo construye, para diferentes valores de los parámetros de entrada. De esta comparación se determina la capacidad que tiene el algoritmo para producir grupos significativos bajo diferentes condiciones iniciales. Algunos ejemplos de este tipo de medidas son FOM (Yeung *et al.*, 2000) y Stability index (Roth *et al.*, 2002), entre otras.

Las medidas internas no necesitan conocer información previa acerca de los objetos a agrupar, así como tampoco necesitan comparar los resultados obtenidos por los algoritmos bajo diferentes condiciones. Las medidas de este tipo evalúan un algoritmo de agrupamiento utilizando solo la información contenida en los grupos que se desean evaluar, como por ejemplo: la compacidad y separación de los grupos, la conectividad de los mismos, entre otros. Algunos ejemplos de este tipo de medidas son los índices David-Bouldin y Silhouette (Halkidi *et al.*, 2001), la medida Λ (Stein *et al.*, 2003), entre otras.

De estos tres tipos de medidas, las más usadas son las medidas externas (Amigó *et al.*, 2009). A continuación se hace un análisis de las características de las medidas externas más usadas para evaluar agrupamientos y se establece la medida que se usará para evaluar los algoritmos propuestos en este capítulo.

Muchas medidas de evaluación externas han sido propuestas en la literatura (Zhao and Karypis, 2001; Larsen and Aone, 1999; Halkidi *et al.*, 2001; Steinbach *et al.*, 2000; Bakus *et al.*, 2002; Rosenberg and Hirschberg, 2007). Estas medidas están basadas en preceptos matemáticos diferentes y tienen diferentes limitaciones y sesgos. Idealmente, se espera que una medida sea imparcial y no tenga preferencias sobre ningún algoritmo de agrupamiento en particular. Sin embargo, esto no siempre se cumple.

Existen diferentes trabajos que han intentado estudiar y comparar las propiedades de las diferentes medidas reportadas, así como han intentado definir un conjunto de restricciones matemáticas que una medida de evaluación ideal debe satisfacer (Dom, 2001; Meila, 2003; Strehl, 2002). Entre los resultados de estos trabajos, se tiene que medidas como Purity (Zhao and Karypis, 2001) y Entropy (Steinbach *et al.*, 2000) tienen preferencias por algoritmos que producen grupos de pocos objetos y que con F1--measure (Larsen and Aone, 1999) los grupos formados al azar no tienen el mínimo valor de calidad.

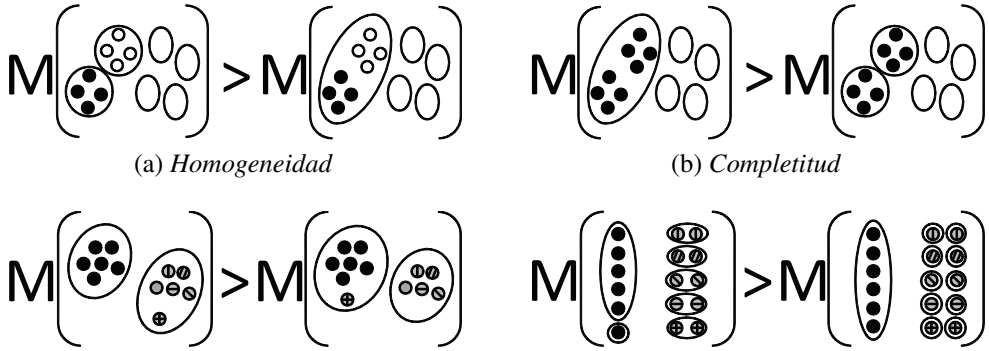
Recientemente, se han propuesto cuatro restricciones matemáticas formales (Amigó *et al.*, 2009). Estas restricciones son intuitivas y ayudan a esclarecer las limitaciones de cada medida de evaluación. Adicionalmente, estas restricciones cubren todos los aspectos propuestos en los estudios previos (Dom, 2001; Meila, 2003; Strehl, 2002).

Las cuatro restricciones formales son:

a) *Homogeneidad*. Esta restricción establece que un mismo grupo no debe contener objetos de diferentes clases (ver Figura 3.1a).

b) *Completitud*. Esta restricción establece que los objetos pertenecientes a una misma clase deben pertenecer a un mismo grupo (ver Figura 3.1b).

c) *Bolsa de ruido*. Esta restricción establece que es preferible obtener un agrupamiento que tenga grupos homogéneos y un grupo ruidoso,



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

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

Figura 3.1: Ilustrando las cuatro restricciones formales.

a obtener un agrupamiento en el cual todos los grupos no sean homogéneos (ver Figura 3.1c).

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 3.1d).

En (Amigó *et al.*, 2009) se mostró que ninguna de las medidas externas más usadas satisface todas las restricciones formales. Las medidas basadas en *cotejo de conjuntos* como Purity (Zhao and Karypis, 2001), Inverse Purity (Zhao and Karypis, 2001) y F1-measure (Larsen and Aone, 1999) no satisfacen las restricciones b y c, mientras que la restricción a solo la satisface la medida Purity. Por otra parte, medidas basadas en *conteo de pares de objetos* como Rand statistic (Halkidi *et al.*, 2001), Jaccard coefficient (Halkidi *et al.*, 2001), Fmeasure (Banerjee *et al.*, 2005) y FM index (Halkidi *et al.*, 2001) no satisfacen las restricciones c y d. Las medidas basadas en el concepto de *entropía* como Entropy (Steinbach *et al.*, 2000), Class Entropy (Bakus *et al.*, 2002), VI measure (Meila, 2003), Mutual Information (Xu *et al.*, 2003), Q_0 (Dom, 2001) y V-measure (Rosenberg and Hirschberg, 2007) no satisfacen la restricción c. Adicionalmente, la medida Class Entropy no satisface la restricción a y la medida Mutual Information no satisface las restricciones b y d. Finalmente, las medidas basadas en *distancia de edición* como la propuesta en (Pantel and Lin, 2002), cumplen las restricciones b y d pero fallan con las restricciones a y c.

Ninguna de las medidas externas mencionadas, ha sido desarrollada, al

menos explícitamente, para evaluar algoritmos de agrupamientos que producen grupos con traslape. Existen trabajos en los que se han utilizado medidas basadas en conteo de pares de objetos, como Fmeasure y Jaccard coefficient, para evaluar agrupamientos traslapados (Banerjee *et al.*, 2005; Pérez-Suárez and Medina-Pagola, 2007; Gago-Alonso *et al.*, 2007; Pérez-Suárez *et al.*, 2009). Otros trabajos, en los cuales se aborda el problema del agrupamiento con traslape, han usado medidas basadas en cotejo de conjuntos como la F1-measure para evaluar sus resultados (Gil-García *et al.*, 2003; Gil-García and Pons-Porrata, 2010). Sin embargo, todas estas medidas no reflejan el hecho de que, en un agrupamiento con traslape, los objetos que compartan n clases en el *ground-truth* deberían compartir n grupos en el agrupamiento.

En (Amigó *et al.*, 2009), se propuso una nueva medida externa para la evaluación de agrupamientos traslapados que, adicionalmente, satisface las cuatro restricciones formales descritas anteriormente. La medida propuesta se llama FBcubed y se calcula utilizando variaciones de las medidas Bcubed precision y Bcubed recall, introducidas en (Bagga and Baldwin, 1998). A continuación se describe la medida FBcubed.

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, denotada por $Bcubed_{pre}(o)$, se calcula de la siguiente forma:

$$Bcubed_{pre}(o) = \frac{\sum_{e \in D(o)} MBcubed_{pre}(o, e)}{|D(o)|}, \tag{3.14}$$

donde $MBcubed_{pre}(o, e)$ es la Bcubed precision del objeto o respecto al objeto e y la misma 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, denotada por $Bcubed_{rec}(o)$, se calcula de la siguiente forma:

$$Bcubed_{rec}(o) = \frac{\sum_{e \in H(o)} MBcubed_{rec}(o, e)}{|H(o)|}, \tag{3.15}$$

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:

$$FBcubed = \frac{2 \cdot Bcubed_{pre} \cdot Bcubed_{rec}}{Bcubed_{pre} + Bcubed_{rec}} \tag{3.16}$$

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.

Una explicación más detallada, así como un caso de estudio y un análisis más profundo acerca de la medida FBcubed, pueden encontrarse en (Amigó *et al.*, 2009).

Con base en el análisis anterior y con el interés de conducir una comparación justa, entre los algoritmos propuestos en esta investigación y aquellos reportados en el estado del arte, se utilizará la medida FBcubed en los experimentos en los que se evalúe la calidad de los grupos formados por cada algoritmo.

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 (Amigó *et al.*, 2009), descrita en la sección 3.4.2. 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.

Inicialmente, todos los algoritmos se ejecutaron sobre la colección AFP, utilizando valores de β en el intervalo [0.10,0.50] y un incremento de 0.01; es decir, los valores de β que se utilizaron fueron β=0.10, β=0.11, β=0.12, ..., 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. Para los algoritmos ISC y Estar se tomó el mayor valor de FBcubed como su mejor evaluación sobre la colección AFP. Como los algoritmos Star, SHC, Gstar, ACONS, ICSD y DClustR son dependientes del orden de procesamiento de los documentos, se repitió veinte veces el experimento anterior, variando el orden de los documentos de la colección. Posteriormente, se calculó el promedio de FBcubed que obtienen estos algoritmos para los valores de β usados. Para cada uno de estos algoritmos, se tomó el mayor valor promedio como su mejor evaluación sobre la colección AFP.

Para determinar la mejor evaluación de cada algoritmo con el resto de las colecciones, se utilizó un procedimiento similar al empleado con la colección AFP. Durante el procesamiento de las colecciones se observó que, para valores de β mayores que 0.50 y menores a 0.10, el valor de calidad de los algoritmos evaluados decreció; por esta razón, no se emplearon en los experimentos valores de β fuera de este rango. Adicionalmente, a pesar de que los algoritmos Star, SHC, Gstar, ACONS, ICSD y DClustR dependen del orden en que se analizan los documentos, la desviación standard del valor de FBcubed obtenido por estos algoritmos fue menor a 0.01, para cada valor de β y para cada colección de prueba. 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.

En la Tabla 3.2 se muestra la mejor evaluación que cada algoritmo logra, respecto a la medida FBcubed, sobre las colecciones de prueba. En la Tabla 3.3 se muestran los valores de β para los cuales los algoritmos obtienen su mejor evaluación. En el caso del algoritmo SHC, los mejores resultados en todas las colecciones se alcanzaron utilizando ϵ=0.05. Adicionalmente, los mejores resultados de SHC con las colecciones AFP, Reuter-Te y Reuter se alcanzaron con HR_{min}=0.35; en las colecciones cacm y cisi los mejores resultados se alcanzaron con HR_{min}=0.20 y, con el resto de las colecciones, los mejores resultados se alcanzaron con HR_{min}=0.40.

Tabla 3.2: 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

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

Como puede verse en la Tabla 3.2, DClustR obtiene mejores valores de calidad, de acuerdo a la medida FBcubed, que el resto de los algoritmos.

Para analizar los resultados de calidad presentados en la Tabla 3.2, se siguió una metodología similar a la utilizada en (Zhao *et al.*, 2005; Gil-García and Pons-Porrata, 2010). Esta metodología consiste en comparar dos a dos los algoritmos de agrupamiento, de acuerdo a los valores de calidad que obtienen sobre todas las colecciones de prueba. En la comparación entre un par de algoritmos A1 y A2, se determina el número de colecciones en las que cada algoritmo mejora/iguala al otro algoritmo y además, se determina la significancia estadística de los resultados. Como resultado de este análisis, En la Tabla 3.4 se muestra la *Matriz de dominancia* y la *Ma-*

Tabla 3.3: Valores de β para los cuales cada algoritmo alcanza sus mejores evaluaciones sobre cada colección

	Algoritmos							
Coll.	Star	ISC	Estar	Gstar	ACONS	ICSD	SHC	DClustR
AFP	0.25	0.10	0.24	0.24	0.27	0.25	0.40	0.21
Reu-Te	0.24	0.10	0.29	0.32	0.34	0.30	0.45	0.20
Reu-Tr	0.20	0.10	0.30	0.30	0.30	0.30	0.50	0.22
Reuter	0.20	0.10	0.25	0.33	0.32	0.30	0.45	0.22
TDT	0.34	0.10	0.36	0.35	0.35	0.30	0.45	0.32
TDT-1	0.33	0.10	0.36	0.35	0.35	0.30	0.45	0.29
TDT-2	0.32	0.10	0.36	0.35	0.35	0.30	0.45	0.30
TDT-3	0.33	0.10	0.36	0.35	0.35	0.30	0.45	0.32
TDT-4	0.30	0.10	0.36	0.35	0.35	0.30	0.45	0.30
TDT-5	0.33	0.10	0.36	0.35	0.35	0.30	0.45	0.32
cacm	0.23	0.17	0.25	0.26	0.26	0.28	0.22	0.26
cisi	0.30	0.18	0.26	0.25	0.27	0.27	0.30	0.21

triz de significancia estadística de los valores de calidad obtenidos por cada algoritmo, sobre las colecciones de prueba.

En la matriz de dominancia, las filas y columnas corresponden con los algoritmos evaluados y cada celda contiene el número de colecciones en las que el algoritmo de la fila obtiene mejores/iguales resultados de calidad que el algoritmos de la columna.

En la matriz de significancia estadística, las filas y columnas corresponden con los algoritmos evaluados y cada celda expresa la significancia estadística que tienen los resultados de calidad del algoritmo de la fila, en comparación con los resultados del algoritmo de la columna. En esta matriz, el símbolo “>>” (“<<”) expresa que los resultados que alcanza el algoritmo de la fila son significativamente mejores (peores) que los alcanzados por el algoritmo de la columna. El símbolo “>” (“<”) expresa que los resultados que alcanza el algoritmo de la fila son mejores (peores) que los que alcanza el algoritmo de la columna, pero que la diferencia no es significativa. El símbolo “–” se usa cuando el algoritmo de la fila es el mismo que el de la columna. Para determinar la significancia estadística entre los valores de FBcubed de los algoritmos 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.

Tabla 3.4: Matriz de dominancia y de significancia estadística de los valores de calidad obtenidos por cada algoritmo sobre las colecciones de prueba

	Matriz de dominancia							
Algoritmos	Star	ISC	Estar	Gstar	ACONS	ICSD	SHC	DClustR
Star	-	12/0	11/0	11/1	11/0	11/0	12/0	0/0
ISC	0/0	-	0/0	0/0	0/0	0/0	1/0	0/0
Estar	1/0	12/0	-	5/5	8/2	6/4	12/0	0/0
Gstar	0/1	12/0	2/5	-	4/6	3/8	12/0	0/0
ACONS	1/0	12/0	4/6	2/6	-	3/8	12/0	0/0
ICSD	1/0	12/0	3/3	1/8	1/8	-	12/0	0/0
SHC	0/0	11/0	0/0	0/0	0/0	0/0	-	0/0
DClustR	12/0	12/0	12/0	12/0	12/0	12/0	12/0	-
	Matriz de significancia estadística							
Algoritmos	Star	ISC	Estar	Gstar	ACONS	ICSD	SHC	DClustR
Star	-	>>	>>	>>	>>	>>	>>	<<
ISC	<<	-	<<	<<	<<	<<	<<	<<
Estar	<<	>>	-	>	>	>	>>	<<
Gstar	<<	>>	<	-	>	>	>>	<<
ACONS	<<	>>	<	<	-	>	>>	<<
ICSD	<<	>>	<	<	<	-	>>	<<
SHC	<<	>>	<<	<<	<<	<<	-	<<
DClustR	>>	>>	>>	>>	>>	>>	>>	-

Como se puede observar en la Tabla 3.4, el algoritmo DClustR supera en todas las colecciones al resto de los algoritmos del estado del arte. Adicionalmente, esta tabla muestra que los valores de calidad que obtiene DClustR son significativamente superiores a los obtenidos por el resto de los algoritmos.

3.4.4. Número de grupos formados

En este experimento, se compara el número de grupos que los algoritmos construyen para cada colección de documentos, usando los parámetros para los cuales cada algoritmo obtiene los mejores valores de calidad (ver sección 3.4.3). La Tabla 3.5 muestra el número de grupos construidos por los algoritmos para cada colección.

Como puede verse en la Tabla 3.5, el algoritmo propuesto obtiene, en casi todas las colecciones, menos grupos que el resto de los algoritmos utili-

Tabla 3.5: Número de grupos construidos por cada algoritmo para las colecciones de prueba. Los valores más pequeños por colección aparecen en negrita

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

zados en la comparación. De hecho, DClustR obtiene respecto al número de grupos que obtienen el resto de los algoritmos, una reducción en el número de grupos que llega a ser del 96.32 %. Es importante resaltar que estos resultados de DClustR se obtienen sin afectar la calidad de los grupos formados. De esta forma, DClustR obtiene agrupamientos que además de tener mejor calidad que los construidos por el resto de los algoritmos, pueden ser analizados con más facilidad pues contienen menos grupos.

Es importante mencionar que la mayoría de los algoritmos, propuestos en el estado del arte, obtienen un gran número de grupos. Cuando se desea descubrir las relaciones o patrones ocultos en los datos, se esperaría obtener una cantidad de grupos que fuera pequeña en comparación con el número de objetos de la colección. De esta forma, el análisis de estos resultados sería más sencillo y fácil de realizar por una persona. Note que, cuando el número de grupos obtenidos es muy alto, como ocurre con la mayoría de los algoritmos propuestos en la literatura, analizar dichos resultados se vuelve tan costoso como analizar la colección; *i.e.*, pierde sentido aplicar alguna técnica de agrupamiento sobre la colección.

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 se calcula como el promedio de grupos en los cuales está incluido cada objeto de la colección (Gil-García and Pons-Porrata, 2010). Si este promedio es mayor que 1, entonces el algoritmo permite construir grupos con traslape. En la Tabla 3.6, se muestra el traslape del conjunto de grupos que construye cada algoritmo para las colecciones de prueba.

Tabla 3.6: 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	DClustR
AFP	1.71	1.65	2.52	2.31	2.48	2.53	2.43	**1.18**
Reu-Te	3.41	1.79	7.40	6.73	6.66	7.64	13.13	**1.40**
Reu-Tr	5.54	1.84	12.14	13.08	12.65	13.32	29.33	**1.56**
Reuter	5.46	1.82	15.92	15.47	15.47	19.25	47.55	**1.53**
TDT	4.81	1.88	59.41	69.43	66.38	70.97	80.74	**1.50**
TDT-1	3.38	1.84	44.22	49.08	46.40	49.63	47.91	**1.43**
TDT-2	3.38	1.80	35.11	37.85	37.46	37.85	39.82	**1.39**
TDT-3	3.81	1.84	42.40	46.08	44.83	46.22	53.79	**1.53**
TDT-4	4.08	1.83	43.34	47.59	46.24	48.72	56.04	**1.45**
TDT-5	3.98	1.88	44.03	51.46	48.44	52.23	59.79	**1.45**
cacm	2.31	1.82	3.46	3.20	3.19	2.72	1.99	**1.26**
cisi	4.12	2.12	7.85	7.49	7.77	7.54	6.98	**1.58**

Como puede verse en la Tabla 3.6, 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 traslape entre los grupos pero, que este traslape sea bajo. Ejemplos de este tipo de aplicaciones son la segmentación de documentos por tópicos (Abella-Pérez and Medina-Pagola, 2010), la organización de documentos web (Zamir and Etziony, 1998; Hammouda and Kamel, 2004), 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 de un algoritmo respecto a una colección es mayor que 1 entonces, el algoritmo produce grupos que tienen un traslape mayor que el existente en la colección. En otro caso, si el traslape relativo es menor a 1 entonces, el algoritmo produce grupos que tienen menos traslape que el que tiene la colección de objetos. 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. Luego, si denotamos por TR el traslape relativo de un algoritmo sobre una colección entonces, 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 pequeño sea el resultado obtenido, más cercano estarán el traslape obtenido por el algoritmo del traslape de la colección y por lo tanto, mejor será el algoritmo.

En la Tabla 3.7 se muestra, para cada colección de prueba, cuán lejos está el traslape construido por cada algoritmo del traslape existente en la colección.

Tabla 3.7: Diferencias entre el traslape del agrupamiento obtenido por el algoritmo del traslape de la colección. Los valores más bajos por colección aparecen en negrita

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

Como puede observarse en la Tabla 3.7, 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. Comportamiento con múltiples cambios

En los experimentos que se presentan en esta sección, se evalúan los algoritmos respecto al tiempo que les toma actualizar el agrupamiento cuando se realizan múltiples cambios sobre la colección. Los experimentos se hicieron sobre la colección más grande utilizada en los experimentos anteriores; *i.e.*, TDT.

Inicialmente, se presentan los experimentos con múltiples adiciones. Posteriormente, se presentan los experimentos con múltiples eliminaciones y por último, se presentan los resultados con múltiples modificaciones.

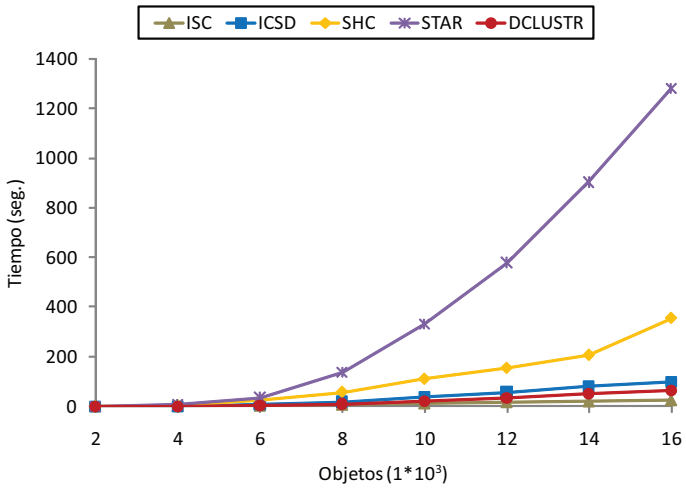
Es importante notar que una comparación justa entre los algoritmos solo es posible si todos usan el mismo valor de β. En otro caso, no se estará seguro si el algoritmo que tiene el mejor comportamiento, debe dicho comportamiento a la estrategia de actualización que emplea o al valor de los parámetros de entrada que utilizaba. Por lo tanto, cualquier valor de β puede usarse en los experimentos siempre y cuando sea el mismo en cada algoritmo. Los resultados que se muestran en esta sección fueron obtenidos utilizando β=0.30. En el caso del algoritmo SHC, se probaron 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}; finalmente, se seleccionaron los valores de ϵ y HR_{min} que producían el mejor tiempo (HR_{min}=0.25 y ϵ =0.05).

Múltiples adiciones

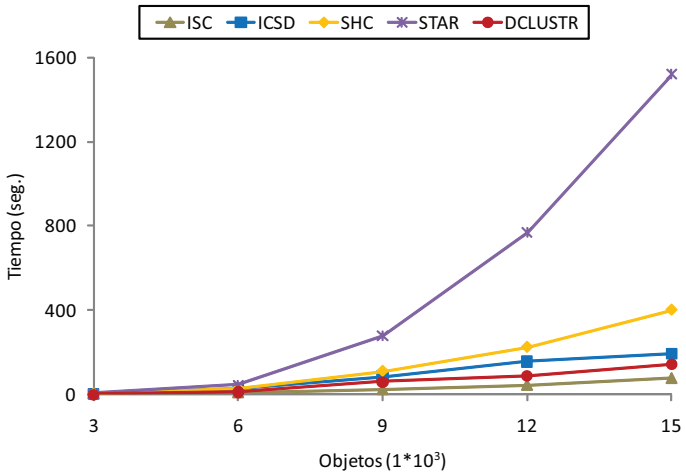
En este experimento se mide el tiempo que tardan los algoritmos Star, ISC, ICSD, SHC y DClustR en actualizar el agrupamiento cada vez que se adicio-

nan N_a documentos a la colección.

Para realizar la comparación, los algoritmos se ejecutaron diez veces para cada valor de N_a, variando cada vez el orden de los documentos de la colección. En la Figura 3.2, se muestra el comportamiento promedio de cada algoritmo para los valores de $N_a = 2000$ (ver Figura 3.2a) y $N_a = 3000$ (ver Figura 3.2b). Se decidió usar estos valores de N_a para tener 8 y 5 bloques de inserciones múltiples respectivamente.



(a) Comportamiento para N_a=2000.



(b) Comportamiento para N_a=3000.

Figura 3.2: Comportamiento de cada algoritmo en el procesamiento de N_a múltiples adiciones sobre TDT.

Como se puede observar en la Figura 3.2, el algoritmo DClustR tiene un mejor comportamiento que los algoritmos Star, SHC y ICSD, cuando se procesan múltiples adiciones sobre la colección TDT. Es válido mencionar que, aunque el algoritmo ISC tiene un comportamiento ligeramente mejor que el de DClustR, este último construye grupos que tienen una mejor calidad que los construidos por ISC (ver sección 3.4.3). Adicionalmente, como se mostró en las secciones 3.4.4 y 3.4.5, DClustR mejora los resultados del algoritmo ISC de acuerdo al número de grupos y al traslape de éstos. Se hicieron otros experimentos considerando otros valores de N_a y β, observándose el mismo comportamiento.

Múltiples eliminaciones

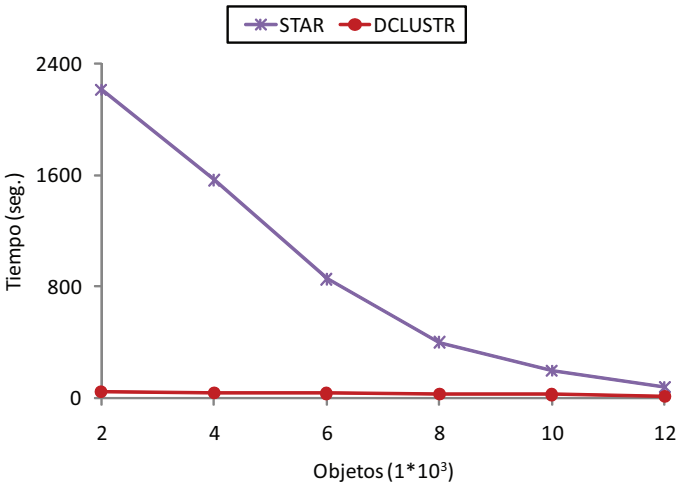
En este experimento se mide el tiempo que tardan los algoritmos Star y DClustR en actualizar el agrupamiento cada vez que se eliminan aleatoriamente N_d documentos de la colección. Es importante mencionar que solo se comparan los resultados contra el algoritmo Star, pues éste es el único algoritmo de agrupamiento traslapado que es capaz de procesar adiciones y eliminaciones. Como valores de N_d se usaron los mismos que se utilizaron para N_a en la sección anterior, así como el mismo valor de β; *i.e.*, β=0.30.

De forma similar a como se realizó para múltiples adiciones, el experimento se repitió diez veces para cada valor de N_d, variando el orden de los documentos de la colección. En la Figura 3.3 se muestra el comportamiento promedio de cada algoritmo para los valores de N_d seleccionados.

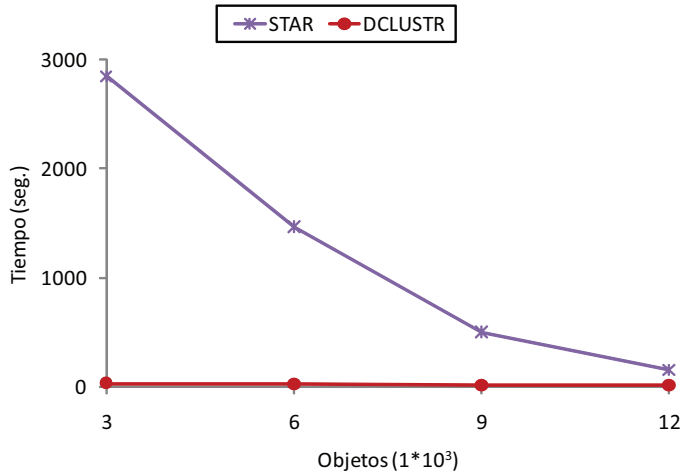
Como se puede observar en la Figura 3.3, el algoritmo DClustR mejora ampliamente los resultados del algoritmo Star en el procesamiento de múltiples eliminaciones de objetos sobre TDT. Se hicieron otros experimentos considerando otros valores de N_d y β, observándose el mismo comportamiento.

Múltiples modificaciones

En este experimento se mide el tiempo que tardan los algoritmos Star y DClustR en el procesamiento de múltiples modificaciones. Solo se comparan los resultados contra el algoritmo Star, pues éste es el único algoritmo



(a) Comportamiento para N_d=2000.



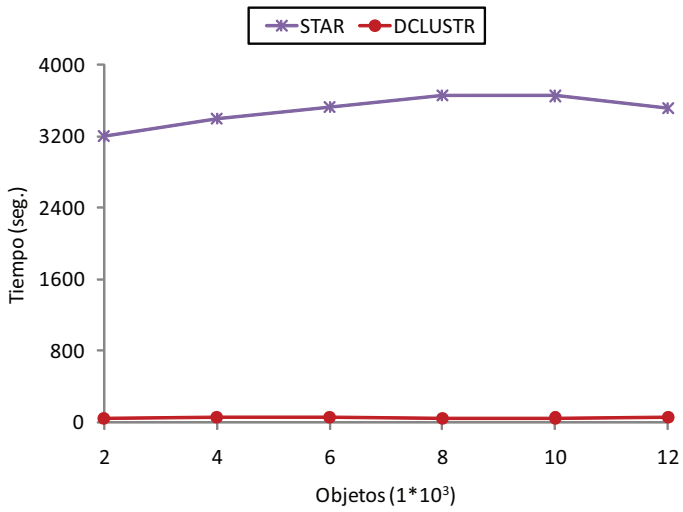
(b) Comportamiento para N_d=3000.

Figura 3.3: Comportamiento de cada algoritmo en el procesamiento de N_d múltiples eliminaciones sobre TDT.

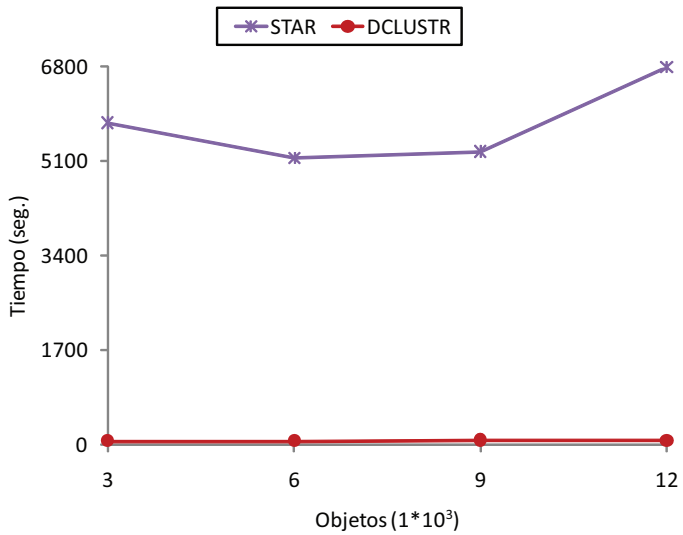
de agrupamiento traslapado que es capaz de procesar adiciones y eliminaciones. Es importante recordar que tanto el algoritmo Star como el DClustR, procesan las modificaciones al procesar una eliminación seguida de una adición. Para construir el agrupamiento previo se utilizó el mismo valor de β utilizado en los experimentos con múltiples adiciones y eliminaciones.

Para realizar el experimento, se mide el tiempo que tardan los algoritmos Star y DClustR en actualizar el agrupamiento cada vez que se N_m documentos son aleatoriamente eliminados y adicionados otra vez pero cambiando el

peso de algunos de sus términos; *i.e.*, modificándolos. Como valores de N_m se usaron los mismos utilizados para N_a y N_d en los experimentos anteriores. Para realizar la comparación, se repitió el mismo experimento diez veces para cada valor de N_m, variando el orden de los documentos de la colección. En la Figura 3.4 se muestra el comportamiento promedio de cada algoritmo para los diferentes valores de N_m.



(a) Comportamiento para N_m=2000.



(b) Comportamiento para N_m=3000.

Figura 3.4: Comportamiento de cada algoritmo en el procesamiento de N_M múltiples modificaciones sobre TDT.

Como se puede observar en la Figura 3.4, el algoritmo DClustR mejora ampliamente los resultados del algoritmo Star en el procesamiento de múltiples modificaciones de objetos sobre TDT. Se hicieron otros experimentos considerando otros valores de N_m y β, observándose el mismo comportamiento.

3.5. Síntesis y conclusiones

En este capítulo se propusieron dos algoritmos de agrupamiento para la formación de grupos con traslape: OClustR y DClustR.

El algoritmo OClustR es estático, representa la colección de objetos por un *grafo de β-semejanza pesado* $\widetilde{G}_\beta$ y construye un agrupamiento traslapado en dos pasos. DClustR es un algoritmo dinámico y se basa en los conceptos introducidos por OClustR para la construcción de los grupos. DClustR introduce además una estrategia para la actualización eficiente del conjunto de grupos, cuando ocurren múltiples adiciones y/o eliminaciones de objetos.

El algoritmo DClustR fue comparado con el resto de los algoritmos traslapados propuestos en el estado del arte, usando varias colecciones estándares. La evaluación experimental se enfocó en comparar los algoritmos respecto a la calidad de los grupos formados, el número de grupos construidos, el traslape entre dichos grupos y el tiempo que tardan los algoritmos para procesar múltiples adiciones y/o eliminaciones de objetos. Es importante mencionar que los algoritmos DClustR y OClustR obtienen los mismos grupos y por lo tanto, los valores de calidad, número de grupos y traslape entre los grupos son los mismos

A partir de estos experimentos, se puede concluir que el algoritmo DClustR obtiene los mejores valores de calidad, de acuerdo a la medida FBcubed, entre todos los algoritmos traslapados del estado del arte. Adicionalmente, a diferencia de los algoritmos utilizados en los experimentos, DClustR construye agrupamientos que tienen menos grupos así como un menor traslape que los agrupamientos construidos por los otros algoritmos.

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

el algoritmo Star. Este algoritmo es el único algoritmo del estado del arte capaz de formar grupos traslapados y de procesar adiciones y/o eliminaciones de objetos. 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.

De todo el análisis anterior, se puede concluir que DClustR representa una mejor opción, que los algoritmos reportados previamente, para enfrentar el problema del agrupamiento traslapado en un contexto dinámico. Con los resultados obtenidos en este capítulo, se consideran como cumplidos los objetivos particulares 1 y 2 de esta tesis doctoral.

Capítulo 4

Algoritmos de agrupamiento jerárquico con traslape

En este capítulo se introducen un nuevo algoritmo de agrupamiento jerárquico, llamado DHClustR, que es dinámico y permite obtener jerarquías traslapadas. La estructura de este capítulo es la que sigue: en la sección 4.1 se presenta, a través de un nuevo algoritmo jerárquico estático llamado HClustR, la estrategia utilizada por DHClustR para formar una jerarquía traslapada. Posteriormente, en la sección 4.2 se presenta el algoritmo DHClustR. En la sección 4.3, se describen los experimentos en los que se evalúa al algoritmo DHClustR y por último, en la sección 4.4, se presentan las conclusiones del capítulo.

4.1. Algoritmo HClustR

En esta sección, se introduce un nuevo algoritmo jerárquico estático. Este algoritmo, nombrado HClustR, permite la construcción de jerarquías traslapadas a través de una estrategia aglomerativa, siendo el nivel superior el más general y el nivel inferior el más específico.

La idea general de HClustR es construir la jerarquía de abajo hacia arriba, utilizando como objetos de cada nivel los grupos construidos del nivel

anterior. Para construir los grupos en un nivel i de la jerarquía, se construye un grafo de β-semejanza pesado $\widetilde{G}_{\beta}^{(i)}$ con los objetos del nivel y, posteriormente, se aplica el algoritmo OClustR (ver sección 3.2). La construcción de la jerarquía se detiene cuando se alcanza un nivel i, en el cual el grafo $\widetilde{G}_{\beta}^{(i)}$ solo está compuesto por vértices aislados.

Hay dos aspectos que determinan el funcionamiento del algoritmo HClustR: (i) la forma en que se representan los objetos de los niveles superiores y (ii) la forma en que se determina la semejanza entre los objetos de los niveles superiores. En esta sección se presentan dos variantes del algoritmo HClustR que emplean diferentes estrategias para la solución de los aspectos (i) y (ii). La primera variante emplea conceptos que han sido utilizados previamente por los algoritmos del estado del arte. Para la definición de la segunda variante se propone, en la sección 4.1.2, una nueva forma de representar a los objetos de los niveles superiores y, en la sección 4.1.3, se propone una nueva forma de determinar la semejanza, entre los objetos de los objetos de los niveles superiores, basada en la representación anterior.

4.1.1. Primera variante de HClustR

La primera variante de HClustR, en lo adelante nombrada como HClustR-v1, representa los objetos de la colección como vectores $V = \{v_1, v_2, \ldots, v_m\}$, donde $v_i \in [0, 1]$. Para calcular la semejanza entre los objetos del primer nivel, HClustR-v1 utiliza la medida del coseno (Berry, 2004). Los objetos de los niveles superiores fueron representados a través del *vector composición* (Zhao and Karypis, 2002) de los grupos del nivel inmediato inferior; la semejanza entre estos objetos se determinó a través de la medida *group-average* (Jain and Dubes, 1988).

Sea $O = \{o_1, o_2, \ldots, o_k\}$ una colección de objetos, $S(o_i, o_j)$ una función de semejanza tal que $\forall o_i, o_j \in O, o_i \neq o_j$, $S(o_i, o_j)$=$S(o_j, o_i)$ y G_1, G_2 dos conjuntos de objetos tales que $G_1 \subseteq O$ y $G_2 \subseteq O$. La semejanza entre los conjuntos G_1 y G_2, de acuerdo a la medida *group-average* (Jain and Dubes, 1988), se denota por $SEM_{ave}(G_1, G_2)$ y se define como:

$$SEM_{ave}(G_1, G_2) = \frac{\sum_{o_i \in G_1, o_j \in G_2} S(o_i, o_j)}{|G_1| \cdot |G_2|} \tag{4.1}$$

donde $|G_1|$ y $|G_2|$ son el número de elementos que contienen los grupos G_1 y G_2 respectivamente.

Si los objetos de la colección O se representan como vectores, si estos vectores están normalizados y se utiliza la medida del coseno (Berry, 2004) como función de semejanza entonces, según (Zhao and Karypis, 2002), la ecuación (4.1) puede reescribirse como:

$$SEM_{ave}(G_1, G_2) = \frac{S(C_{G_1}, C_{G_2})}{|G_1| \cdot |G_2|},$$

luego:

$$SEM_{ave}(G_1, G_2) = \frac{\langle C_{G_1}, C_{G_2} \rangle}{|G_1| \cdot |G_2|}, \tag{4.2}$$

donde $\langle C_{G_1}, C_{G_2} \rangle$ es el producto escalar entre C_{G_1} y C_{G_2}; C_{G_1} y C_{G_2} son los *vectores de composición* de los grupos G_1 y G_2 respectivamente. El vector composición de G_1 se obtiene sumando los vectores de cada uno de los objetos del grupo G_1 (Zhao and Karypis, 2002); C_{G_2} se define de la misma forma que C_{G_1}. Sean G_1 y G_2 dos grupos obtenidos por HClustR en el nivel $i > 1$ y β el umbral de semejanza. Los grupos G_1 y G_2 son semejantes, de acuerdo a la fórmula (4.2), si se cumple que $SEM_{ave}(G_1, G_2) \geq \beta$.

El pseudocódigo de la primera variante del algoritmo HClustR se muestra en el Algoritmo 4.1. En este algoritmo, $\widetilde{G}_\beta^{(i)} = \langle V^{(i)}, \widetilde{E}_\beta^{(i)}, w \rangle$ es el grafo de β-semejanza pesado que representa a los objetos del nivel i, SC el conjunto de grupos del nivel actual y CV el conjunto de vectores de composición asociado a SC. Adicionalmente, HC es la jerarquía de grupos que construye en el algoritmo y HG es la jerarquía de grafos asociados a cada nivel.

Algoritmo 4.1: Algoritmo HClustR - variante 1

Input: $O = \{o_1, o_2, \ldots, o_n\}$ - colección de objetos, β - umbral de semejanza
Output: HC - jerarquía de grupos, HG - jerarquía de grafos

1 "Construir $\widetilde{G}_\beta^{(1)} = \langle V^{(1)}, \widetilde{E}_\beta^{(1)}, w \rangle$ a partir de la colección O y el umbral β";
2 $i := 1$;
3 **repeat**
4 "Marcar los vértices en $\widetilde{G}_\beta^{(i)}$ como *satélites*";
5 "Calcular la *relevancia* de cada vértice en $V^{(i)}$";
6 $SC := \emptyset$;
7 $CV := \emptyset$;
8 `OClustR` ($\widetilde{G}_\beta^{(i)}$,$SC$);
9 "Construir CV a partir de SC";
10 $HC[i] := SC$;
11 $HG[i] := \widetilde{G}_\beta^{(i)}$;
12 $i++$;
13 "Construir $\widetilde{G}_\beta^{(i)} = \langle V^{(i)}, \widetilde{E}_\beta^{(i)}, w \rangle$ a partir de la colección CV y de β";
14 **until** $\left|\widetilde{E}_\beta^{(i)}\right| = 0$;

Como se observa en el Algoritmo 4.1, se construye el grafo de β-semejanza pesado que representa a los objetos de cada nivel (pasos 1 y 13) como parte del código de HClustR. El umbral de semejanza y los objetos del primer nivel se reciben como parámetros y, en el paso 9, se construyen los vectores de composición de los grupos. El umbral de semejanza y los objetos del primer nivel se utilizan para construir $\widetilde{G}_{\beta}^{(1)}$ en el paso 1; el umbral de semejanza y los vectores de composición son utilizados para construir $\widetilde{G}_{\beta}^{(i)}$ en el paso 13. Dado que este grafo se construye como parte de HClustR, ya no es necesario que se construya como parte del algoritmo OClustR; por lo tanto, ahora OClustR tiene como parámetros de entrada el grafo $\widetilde{G}_{\beta}^{(i)}$ sobre el cual tiene que formar el agrupamiento. Adicionalmente, se incluyó también como parte del algoritmo HClustR, marcar todos los vértices del nivel como *satélites* y calcular de la relevancia de estos vértices (ver pasos 4 y 5 respectivamente); luego, no es necesario hacer estas dos acciones en el algoritmo OClustR. Es importante mencionar que las modificaciones anteriores no afectan la complejidad del algoritmo OClustR (ver sección 3.2.3).

Aunque la estrategia utilizada por HClustR es similar a la utilizada por el algoritmo DHS (Gil-García and Pons-Porrata, 2010), existen diferencias entre ambas. En la estrategia que utiliza la primera version de HClustR solo hay que construir el grafo de β-semejanza pesado $\widetilde{G}_{\beta}$. Por otra parte, DHS tiene que construir dos grafos, un grafo de β-semejanza G_{β} para representar a los objetos del nivel y un grafo de máxima β-semejanza $G_{max-\beta}$ para construir los grupos. Note que la construcción de este segundo grafo implica tiempo adicional de procesamiento. Otra diferencia radica en el algoritmo de agrupamiento que se aplica para construir los grupos del nivel. HClustR utiliza el algoritmo OClustR para obtener el agrupamiento en cada nivel y el algoritmo DHS utiliza una variante del algoritmo Star.

4.1.2. Representación de los objetos en los niveles superiores

Como se mencionó anteriormente, el algoritmo DHS y la primera variante del algoritmo HClustR, representan los objetos de los niveles superiores a través del vector composición. Como el agrupamiento obtenido en cada nivel puede ser traslapado, para la formación del vector de composición de un grupo G del nivel i, no se pueden utilizar los vectores de composición de los

sub-grupos del nivel $i-1$ que están contenidos en G. Luego, dicho vector tiene que ser construido utilizando los vectores de los objetos contenidos en el grupo G. Esto aumenta el tiempo de procesamiento del algoritmo y por tanto es un aspecto negativo a resolver. Para solucionar este aspecto se emplearán conceptos utilizados en el algoritmo OClustR.

La idea al representar un grupo G_i por el vector de composición, es hacer que todos los elementos de G_i participen en la decision de si algún otro grupo es semejante o no a G_i. La representación que se propone en esta sección está enfocada a reducir el número de elementos de G_i que participan en dicha decision.

Sea $G = \{v_1, v_2, \ldots, v_m\}$ con $v_j \in V^{(i)}, \forall j = 1..m$, un grupo formado por OClustR en el nivel i. Supóngase, sin perder generalidad, que v_1 es el vértice marcado como centro, que determina el grupo G. Sea $P = \{p_1, p_2, \ldots, p_t\} \subseteq (G \setminus \{v_1\})$ con $p_j \in V^{(i)}, \forall j = 1..t$, el conjunto de vértices prescindibles por los cuales se adjuntaron vértices al ws-grafo determinado por v_1 (ver sección 3.2.2). El *núcleo* del grupo G, denotado por $Core(G)$, es la tupla formada por el vértice v_1 y todos los vértices de P; es decir, $Core(G)$=$(v_1, p_1, p_2, \ldots, p_t)$. La segunda variante del algoritmo HClustR, considera que los objetos de los niveles superiores están representados por los núcleos de los grupos del nivel inmediato inferior.

Cuando se representa el grupo G a través de su núcleo, se está estableciendo que solo los objetos del núcleo determinarán si G es semejante o no a otro grupo. El núcleo de un grupo se puede construir durante la formación del grupo, sin afectar la complejidad del algoritmo OClustR. Note que, en la definición del núcleo de un grupo, a diferencia del vector composición, no se exige que los objetos se representen por vectores de $\Re^n$.

4.1.3. Cálculo de la semejanza en los niveles superiores

Como ya se explicó en la sección 2.2, con el objetivo de poder aplicar la fórmula (4.2) para acelerar el cálculo de la semejanza entre los objetos de los niveles superiores, el algoritmo DHS representa a los grupos a través de su vector composición y utiliza como función de semejanza a la medida del coseno. Estas restricciones limitan el campo de aplicación del algoritmo DHS pues, en principio, obliga a que los objetos estén descritos por rasgos

numéricos. La primera versión del algoritmo HClustR utiliza también la fórmula (4.2) y por lo tanto, sufre de la misma limitación que DHS. Es importante observar que, en caso de que estos algoritmos no utilicen la fórmula (4.2), el cálculo de la semejanza entre dos grupos G_1 y G_2 implicaría el cálculo de $|G_1| \cdot |G_2|$ semejanzas; de esta forma, aumentaría el tiempo de procesamiento de ambos algoritmos.

Para saber si dos grupos G_1 y G_2 son semejantes, de acuerdo a la medida *group-average*, primero se calcula $SEM_{ave}(G_1, G_2)$ y si $SEM_{ave}(G_1, G_2) \geq \beta$, entonces G_1 y G_2 son semejantes y su valor de semejanza es precisamente $SEM_{ave}(G_1, G_2)$. La estrategia que se propone en esta sección para saber si dos grupos G_1 y G_2 son semejantes, utiliza solamente a los objetos que pertenecen a los núcleos de ambos grupos.

Sea $\widetilde{G}_\beta^{(i)} = \langle V^{(i)}, \widetilde{E}_\beta^{(i)}, w \rangle$ el grafo que representa a los objetos del nivel i. Sean v y u los vértices marcados como *centro* que determinan a los grupos G_1 y G_2 respectivamente. Sea además $Core(G_1) = \{\mathrm{v}, p_1, p_2, \ldots, p_t\}$ y $Core(G_2) = \{\mathrm{u}, q_1, q_2, \ldots, q_d\}$ los núcleos de los grupos G_1 y G_2 respectivamente. Entre los vértices de $Core(G_1)$ y $Core(G_2)$ pueden existir tres tipos de semejanzas:

i) Semejanza *centro-centro* (CC). Es la semejanza entre los vértices v y u. Esta semejanza existe cuando hay una arista entre los vértices v y u; *i.e.*, $(v, u) \in \widetilde{E}_\beta^{(i)}$.

ii) Semejanza *centro-prescindible* (CP). Es la semejanza entre v y los vértices prescindibles de $Core(G_2)$ o entre u y los vértices prescindibles de $Core(G_1)$. Esta semejanza existe cuando hay al menos una arista entre v y los prescindibles de $Core(G_2)$ o entre u y los prescindibles de $Core(G_1)$.

iii) Semejanza *prescindible-prescindible* (PP). Es la semejanza entre los vértices prescindibles de $Core(G_1)$ y los vértices prescindibles de $Core(G_2)$. Esta semejanza existe cuando hay al menos una arista entre los prescindibles de $Core(G_1)$ y los prescindibles de $Core(G_2)$.

Para determinar si G_1 y G_2 son semejantes se realizan tres pasos. En el primer paso se determina el número de semejanzas, de cada tipo, que existen entre los objetos de $Core(G_1)$ y $Core(G_2)$. Posteriormente, en el segundo paso se determina el valor de semejanza de tipo A, B y C que existe entre

los grupos G_1 y G_2. Estos valores de semejanza se determinan a partir de las siguientes condiciones:

i) Semejanza de *tipo-A*. El valor de semejanza *tipo-A* que existe entre G_1 y G_2, denotada por $S_A(G_1, G_2)$, es el valor de semejanza CC que existe en $\widetilde{G}_\beta^{(i)}$ entre los vértices $v \in Core(G_1)$ y $u \in Core(G_2)$; *i.e.*, $w(v, u)$. Si $(v, u) \notin \widetilde{E}_\beta^{(i)}$, entonces $S_A(G_1, G_2) = 0$.

ii) Semejanza de *tipo-B*. Sea $C_1 \subseteq \{q_1, q_2, \ldots, q_d\}$, los vértices prescindibles de $Core(G_2)$ con los cuales v tiene semejanzas CP y $C_2 \subseteq \{p_1, p_2, \ldots, p_t\}$, los vértices prescindibles de $Core(G_1)$ con los cuales u tiene semejanzas CP. Sea $C_{CP}(G_1, G_2)$ el número de semejanzas CP que existen entre los vértices de $Core(G_1)$ y de $Core(G_2)$. Sea $S_{CP}(G_1, G_2) = \sum_{x \in C_1} w(v, x) + \sum_{y \in C_2} w(u, y)$ la sumatoria de los pesos de las aristas formadas entre los vértices de $Core(G_1)$ y $Core(G_2)$ que tienen semejanza CP. El valor de semejanza *tipo-B* que existe entre G_1 y G_2, denotada por $S_B(G_1, G_2)$, se define como sigue:

$$S_B(G_1, G_2) = \begin{cases} \frac{S_{CP}(G_1,G_2)}{C_{CP}(G_1,G_2)} & \text{si } |Core(G_1)| + |Core(G_2)| > 2 \\ & \text{y} \\ & \frac{C_{CP}(G_1,G_2)}{|Core(G_1)|+|Core(G_2)|-2} \geq \beta \\ 0 & \text{en otro caso} \end{cases},$$

donde, $|Core(G_1)| + |Core(G_2)| - 2$ es el número total de semejanzas CP que pueden existir entre G_1 y G_2.

iii) Semejanza de *tipo-C*. Sea $C_3 = \{(p, q) \mid p \in \{p_1, p_2, \ldots, p_t\} \wedge q \in \{q_1, q_2, \ldots, q_d\}\}$ el conjunto de pares de vértices prescindibles de $Core(G_1)$ y $Core(G_2)$ que tienen semejanza PP. Sea $C_{PP}(G_1, G_2) = |C3|$ el número de semejanzas PP que existen entre los vértices de $Core(G_1)$ y de $Core(G_2)$. Sea $S_{PP}(G_1, G_2) = \sum_{(x,y) \in C_3} w(x, y)$ la sumatoria de los pesos de las aristas formadas entre los vértices de $Core(G_1)$ y $Core(G_2)$ que tienen semejanza PP. El valor de semejanza *tipo-C* que existe entre G_1 y G_2, denotada por $S_C(G_1, G_2)$, se define como sigue:

$$S_C(G_1, G_2) = \begin{cases} \frac{S_{PP}(G_1,G_2)}{C_{PP}(G_1,G_2)} & \text{si } |Core(G_1)| > 1 \text{ y } |Core(G_2)| > 1 \\ & \text{y} \\ & \frac{C_{PP}(G_1,G_2)}{(|Core(G_1)|-1)\cdot(|Core(G_2)|-1)} \geq \beta \\ 0 & \text{en otro caso} \end{cases},$$

donde, $(|Core(G_1)|-1)\cdot(|Core(G_2)|-1)$ es el número total de semejanzas PP que pueden haber entre G_1 y G_2.

El valor de semejanza entre G_1 y G_2, denotada por $S_T(G_1, G_2)$, se calcula en el tercer paso como sigue:

$$S_T(G_1, G_2) = \frac{S_A(G_1, G_2) + S_B(G_1, G_2) + S_C(G_1, G_2)}{3} \tag{4.3}$$

Con base en lo presentado anteriormente, se dirá que dos grupos G_1 y G_2 son semejantes si se cumple que $S_T(G_1, G_2) > 0$; en otro caso, se dirá que los grupos no son semejantes.

Es importante mencionar que el criterio propuesto para determinar la semejanza entre los objetos de los niveles superiores, no depende del espacio de representación de los objetos ni de la medida de semejanza utilizada en el primer nivel. Otro aspecto a favor del criterio introducido, es que reutiliza las semejanzas calculadas en el nivel anterior y por lo tanto, el cálculo de la semejanza entre dos objetos resulta menos costoso.

4.1.4. Segunda variante de HClustR

La segunda variante de HClustR no depende del espacio de representación de los objetos de la colección y representa a los objetos de los niveles superiores a través del núcleo de los grupos del nivel inmediato inferior. Para calcular la semejanza entre los objetos de los niveles superiores y formar los grafos correspondientes a dichos niveles se utiliza la medida de semejanza S_T, propuesta en la sección 4.1.3. Para el cálculo de la semejanza entre los objetos de primer nivel de la jerarquía se puede utilizar cualquier medida de semejanza, siempre y cuando ésta sea simétrica.

El pseudocódigo de la segunda variante del algoritmo HClustR es similar al que se mostró en el Algoritmo 4.1. La única diferencia radica en que, en esta segunda variante, en el paso 9 se calcula el núcleo asociado a cada grupo de SC en lugar de su vector composición.

4.1.5. Análisis de la complejidad computacional

Las diferencias entre las dos variantes de HClustR radican en: (i) la forma de representar a los objetos de los niveles $i > 1$ y (ii) en la forma de determinar al semejanza entre estos objetos.

A pesar del cambio en la forma de determinar la semejanza entre los objetos de los niveles $i > 1$, en ambas versiones se tienen que visitar todos los objetos del nivel para construir el grafo correspondiente a dicho nivel. Luego, la diferencia en la forma de determinar la semejanza no marca diferencia en la complejidad de las dos versiones de HClustR. La representación de cada objeto se construye en el paso 9 y sí hay diferencias en la complejidad de ambas versiones. Por lo tanto, para determinar la complejidad computacional de las dos versiones del algoritmo HClustR, se analizará cada uno de los pasos del Algoritmo 4.1 y se establecerá las diferencias que pueden existir de acuerdo a la complejidad del paso 9.

Sea n el número de objetos de la colección $O = \{o_1, o_2, \ldots, o_n\}$; *i.e.*, $|O| = n$. Para construir $\widetilde{G}_\beta^{(1)}$ en el paso 1, se debe calcular la semejanza entre todo par de objetos de O. Por lo tanto, el número de operaciones que se realiza en este paso es $T_1(n) = n^2$; así, $T_1(n)$ es $O(n^2)$. Durante este paso se puede calcular, usando la expresión (3.4), la semejanza intra-grupo aproximada del ws-grafo inducido por cada vértice de $\widetilde{G}_\beta^{(1)}$, sin que se afecte $T_1(n)$.

El número de operaciones realizadas en los pasos 3-14 depende de dos factores: la cantidad de operaciones realizadas en los pasos 4-13 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 el paso 4 es $T_4(n_i) = n_i$; luego, $T_4(n_i)$ es $O(n_i)$. A partir del análisis presentado en la sección 3.2.3, se puede concluir que el número de operaciones realizadas en los pasos 5 y 8 es $T_5(n_i) = n_i^2$ y $T_8(n_i) = n_i^2$ respectivamente. Por tanto, $T_5(n_i)$ es $O(n_i^2)$ y $T_8(n_i)$ es $O(n_i^2)$.

En el paso 9 se construyen los vectores de composición o los núcleos de los grupos formados en el nivel, en dependencia si es la primera o la segunda variante de HClustR respectivamente. Dado que los grupos que se obtienen en cada nivel pueden ser traslapados, el vector composición de un grupo G del nivel i no se puede construir a partir de los vectores de composición de los sub-grupos del nivel $i-1$ que forman a G, sino a partir de

los vectores de los objetos contenidos en G. Luego, en la primera variante de HClustR, en lo que sigue referida como HClustR-v1, para formar el vector composición de cada grupo G es necesario recorrer, en el peor caso, todos los objetos de la colección. Por otra parte, los núcleos de los grupos en la segunda variante de HClustR, en lo que sigue referida como HClustR-v2, se construyen visitando solamente los objetos de cada nivel. Luego, si todos los objetos del nivel fueron marcados como centros entonces, en el peor caso, el número de operaciones que se realiza en el paso 9 en HClustR-v1 es $T_9(n_i) = n_i \cdot n$. De forma similar, el número de operaciones que se realiza en el paso 9 en HClustR-v2 es $T_9(n_i) = n_i^2$. Como n_i es siempre menor o igual que n, se puede asumir que en el peor caso el número de operaciones realizadas en cualquiera de las dos variantes de HClustR es $T_9(n_i) = n_i \cdot n$.

Como se mencionó en la descripción del algoritmo HClustR, el número de grupos de un nivel es el número de objetos del nivel siguiente. Por lo tanto, si todos los objetos del nivel fueron marcados como centros entonces, en el peor caso, el número de operaciones que se realiza en el paso 13 es $T_{13}(n_i) = n_i^2$; luego, $T_{13}(n_i)$ es $O(n_i^2)$.

Del análisis anterior se tiene que el número de operaciones realizadas en los pasos 4-13 es $T_{4-13}(n_i) = T_4(n_i)+T_5(n_i)+T_8(n_i)+T_9(n_i)+T_{13}(n_i)$. Por la regla de la suma se tiene que $T_{4-13}(n_i)$ es $O(T_4(n_i), T_5(n_i), T_8(n_i), T_9(n_i), T_{13}(n_i))$; luego, $T_{4-13}(n_i)$ es $O(n_i \cdot n)$.

Como se mencionó anteriormente, el número máximo de vértices que se puede marcar como centro 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 aislados. Este caso ocurre cuando el número de vértices marcados como centro 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 aislados. En otro caso, si estos vértices no fueran aislados, entonces estarían cubiertos y formarían ws-grafos que tienen sus satélites cubiertos por al menos otro vértice; luego, esto implica que dichos vértices son prescindibles y debieron ser eliminados por el algoritmo OClustR. Sea t el número de niveles construidos y $n_1, n_2, \ldots, n_t$ el número de objetos de cada nivel. Como conclusión de este análisis se tiene que, en el peor de los casos, $t = \log_2 n$ y que $n_i = \frac{n_{i-1}}{2}, i = 2..t$.

Con base en el análisis anterior, el número de operaciones realizadas en los pasos 3-14 es:

$$T_{3-14}(n) = \sum_{i=1}^{\log_2 n} T_{4-13}(n_i) = n_1 \cdot n + \frac{n_1 \cdot n}{2} + \frac{n_1 \cdot n}{4} + \frac{n_1 \cdot n}{8} + \ldots + \frac{n_1 \cdot n}{2^{\log_2 n}}$$

conociendo que $n_1 = n$ se tiene que:

$$T_{3-14}(n) = n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \frac{n^2}{8} + \ldots + \frac{n^2}{2^{\log_2 n}} \tag{4.4}$$

Como se puede observar en (4.4), $T_{3-14}(n)$ es una progresión geométrica cuya suma de sus n primeros términos es:

$$T_{3-14}(n) = \frac{a_n \cdot r - a_1}{r - 1},$$

donde $r = \frac{1}{2}$ es la como razón de la progresión, $a_n = \frac{n^2}{2^{\log_2 n}}$ el último término y $a_1 = n^2$ el primer término. Desarrollando la fórmula anterior se tiene que:

$$T_{3-14}(n) = \frac{\frac{n^2}{2^{\log_2 n}} \cdot \frac{1}{2} - n^2}{\frac{1}{2} - 1} = \frac{n^2 \cdot \left(\frac{1}{2^{\log_2 n}} \cdot \frac{1}{2} - 1\right)}{-\frac{1}{2}} = n^2 \cdot \left(2 - \frac{1}{2^{\log_2 n}}\right)$$

$$T_{3-14}(n) = n^2 \cdot \left(2 - \frac{1}{n}\right),$$

y por tanto:

$$T_{3-14}(n) = 2 \cdot n^2 - n \tag{4.5}$$

A partir de (4.5) se puede concluir que $T_{3-14}(n)$ es $O(n^2)$. Finalmente, el número de operaciones realizadas en el Algoritmo 4.1 es $T_t(n) = T_1(n) + T_{3-14}(n)$. Por la regla de la suma, se tiene que $T_t(n)$ es $O(T_1(n), T_{3-14}(n))$ y por lo tanto, $T_t(n)$ es $O(n^2)$. Luego, la complejidad computacional del algoritmo HClustR, en cualquiera de sus dos variantes, es $O(n^2)$.

4.2. Algoritmo dinámico DHClustR

En esta sección se introduce un nuevo algoritmo de agrupamiento jerárquico, nombrado DHClustR, que es dinámico y permite obtener jerarquías traslapadas. DHClustR utiliza la idea general propuesta por HClustR para la formación de jerarquías traslapadas (ver sección 4.1) aunque, a diferencia de

HClustR, el algoritmo DHClustR utiliza a DClustR para actualizar el agrupamiento de cada nivel de la jerarquía. Adicionalmente, DHClustR introduce una estrategia para la actualización de la jerarquía cuando ocurren múltiples adiciones y/o eliminaciones de objetos en la colección.

4.2.1. Actualización de la jerarquía

En la sección 3.3, se explicó cómo las adiciones y/o eliminaciones de objetos afectan el cubrimiento de una colección agrupada por el algoritmo DClustR. Luego, el agrupamiento del primer nivel de la jerarquía formada por HClustR puede actualizarse, cuando ocurren adiciones y/o eliminaciones, utilizando al algoritmo DClustR.

Como se explicó en la sección 4.1, los grupos obtenidos en un nivel son los objetos agrupados en el nivel siguiente. Cuando el agrupamiento del nivel base de la jerarquía es actualizado, se eliminan y/o adicionan grupos; por lo tanto, se deben de eliminar y/o adicionar objetos en el nivel siguiente y consecuentemente, se debe actualizar el agrupamiento de dicho nivel. Así, cada vez que se actualiza el agrupamiento de un nivel 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 de HClustR.

De forma similar al algoritmo HClustR, se tiene dos variantes de DHClustR, en dependencia de si se utiliza la estrategia de HClustR-v1 o si se utiliza la estrategia de HClustR-v2. En adelante, cuando se hable de DHClustR-v1 se estará haciendo referencia a la variante de DHClustR que sigue la estrategia de HClustR-v1. En otro caso, si se habla de DHClustR-v2 entonces, se estará haciendo referencia a la variante de DHClustR que sigue la estrategia de HClustR-v2. La estrategia de actualización explicada anteriormente es la empleada tanto para DHClustR-v1 como para DHClustR-v2.

Un aspecto que marca una diferencia entre DHClustR-v1 y DHClustR-v2, es el tratamiento que se da a los grupos *modificados*. A continuación, se analizará qué tratamiento se le da a este tipo de grupos en DHClustR-v1 y luego, qué tratamiento se le da a estos grupos en DHClustR-v2.

Sea v el vértice, marcado como *centro*, que determina un grupo G en el nivel i de la jerarquía formada por DHClustR-v1. El grupo G se considera como modificado si se adicionan o eliminan vértices adyacentes o vértices ligados a v; *i.e.*, si cambian los conjuntos $v.Adj$ o $v.Linked$. Note que, como DHClustR-v1 utiliza todos los elementos de G para determinar la semejanza de G con otros grupos, cualquier cambio en G hace que éste se marque como modificado.

En el algoritmo DHClustR-v2, para saber si dos grupos del nivel i son semejantes, se utilizan los objetos de los núcleos de ambos grupos. Por lo tanto, para saber si un grupo G ha sido modificado y se debe recalcular su semejanza con otros grupos, hay que analizar los cambios en los objetos de $Core(G)$.

Sea v el vértice, marcado como *centro*, que determina un grupo G en el nivel i de la jerarquía formada por DHClustR-v2. Sea además $Core(G) = \{v, p_1, p_2, \ldots, p_t\}$ el núcleo del grupo G luego de actualizar el agrupamiento del nivel i. Para determinar si G se modificó o no se analizan dos casos. En el primer caso, supóngase que el grupo G tenía semejanza de *tipo-B* o de *tipo-C* con al menos un grupo G'. Si el grupo G' no fue eliminado luego de la actualización, entonces G se considera modificado si se adicionaron o eliminaron vértices prescindibles a $Core(G)$. En el segundo caso, si G no tenía semejanza de *tipo-B* o de *tipo-C* con ningún otro grupo, se procede de acuerdo a las siguientes condiciones:

i) Si se adicionó a $Core(G)$ algún vértice prescindible p que es adyacente a algún otro vértice marcado como *centro*, entonces G se considera como modificado.

ii) Sea G' un grupo que ya existía en el agrupamiento antes de la actualización y $Core(G')$ el núcleo del mismo. Si se adicionó a $Core(G)$ algún vértice prescindible p que es adyacente al menos a un vértice prescindible p' de $Core(G')$, entonces el grupo G se considera como modificado. Si p' es un vértice prescindible que se adicionó a $Core(G')$ luego

de actualizar el agrupamiento del nivel, entonces el grupo G' también se considera como modificado.

Una vez que se detecta, tanto en DHClustR-v1 como en DHClustR-v2, que un grupo G del nivel i se modificó, se elimina del nivel $i+1$ el objeto que representaba al antiguo grupo G. Posteriormente, el objeto que representa actualmente a G se incorpora a la lista de objetos a adicionar en el nivel siguiente. Es importante mencionar que tanto en DHClustR-v1 como en DHClustR-v2, el proceso de análisis de grupos modificados puede realizarse como parte del algoritmo DClustR sin modificar la complejidad computacional de este último.

El pseudocódigo del algoritmo DHClustR se muestra en el Algoritmo 4.2.

Algoritmo 4.2: Algoritmo DHClustR

Input: $O = \{o_1, o_2, \ldots, o_n\}$ - colección de objetos, HG - jerarquía de grafos, β - umbral de semejanza, R^+ - conjunto de objetos adicionados, R^- - conjunto de objetos eliminados
Output: O - colección de objetos actualizada, HG - jerarquía de grafos actualizada, HC - jerarquía de grupos

```
1  i := 1;
2  G̃_β^(i) := HG[i];
3  UpdGraph(O,G̃_β^(i),β,R+,R−,M);
4  repeat
5      if i ≠ 1 and G̃_β^(i) es el tope de la jerarquía then
6          Z := {v | v ∈ V^(i) ∧ |v.Adj| = 0};
7          M := M ∪ Z;
8      end
9      SC := ∅;
10     DClustR (G̃_β^(i),SC,R+,R−);
11     HC[i] := SC;
12     HG[i] := G̃_β^(i);
13     i++;
14     O' := V^(i);
15     G̃_β^(i) := HG[i];
16     "Actualizar los núcleos o los vectores de composición de O' a partir de SC";
17     UpdGraph(O',G̃_β^(i),β,R+,R−,M);
18 until |Ẽ_β^(i)| = 0 ;
19 if G̃_β^(i) no es el tope de la jerarquía then "Eliminar niveles siguientes";
```

Para actualizar el agrupamiento en cada nivel, como se mencionó anteriormente, se utiliza el algoritmo DClustR (ver Algoritmo 3.2). Para procesar las adiciones y/o eliminaciones de objetos en cada nivel, se utiliza el procedimiento *UpdGraph* (ver Algoritmo 3.3). Es válido aclarar que la medida de semejanza que se utiliza en el procedimiento *UpdGraph*, depende del nivel

que se esté procesando y de la version de DHClustR. Si se está procesando un nivel diferente del primero, se utiliza la medida de semejanza S_T (ver sección 4.1.3) si es la version DHClustR-v2 o la medida group-average si es la versión DHClustR-v1. Para el cálculo de la semejanza en el primer nivel se utiliza la medida del coseno si es la versión DHClustR-v1; en otro caso, si es la versión DHClustR-v2, se utiliza la medida de semejanza definida para comparar objetos de la colección.

Como se puede observar en el Algoritmo 4.2, se incluyó el llamado al procedimiento *UpdGraph* como parte de DHClustR (ver pasos 3 y 17). La información necesaria para invocar al procedimiento *UpdGraph* se recibe como parámetro o se calcula en los pasos 12-16. Dado que del procedimiento *UpdGraph* se hace en DHClustR, ya no es necesario invocar a dicho procedimiento como parte del algoritmo DClustR; por lo tanto, ahora DClustR tiene como parámetros de entrada el grafo $\widetilde{G}_{\beta}^{(i)}$ cuyo agrupamiento tiene que actualizar. Adicionalmente, se adicionaron como parámetros de salida del algoritmo DClustR los conjuntos R^{+} y R^{-}. Estos dos conjuntos representan a los objetos que serán adicionados y/o eliminados en el próximo nivel. Es importante mencionar que ninguna de estas modificaciones afecta la complejidad del algoritmo DClustR (ver sección 3.3.2).

Como puede notarse en el Algoritmo 4.2, DHClustR asume que existe una estructura HG que contiene a los grafos $\widetilde{G}_{\beta} = \langle V, \widetilde{E}_{\beta}, w \rangle$ que representan a la colección de objetos en cada nivel. Si no hubiera un agrupamiento previo y es la primera vez que se va a procesar a la colección O, entonces HG es vacío; de esta forma, DHClustR puede procesar una colección de objetos sin la existencia de un agrupamiento previo.

4.2.2. Análisis de la complejidad computacional

Para determinar la complejidad computacional del algoritmo DHClustR, se analizarán cada uno de sus pasos. Sea n el número de vértices de $\widetilde{G}_{\beta}^{(1)}$; *i.e.*, $\left|V^{(1)}\right| = n$. A partir del análisis presentado en la sección 3.3.2, se tiene que el número de operaciones realizadas en el procedimiento *UpdGraph*, para actualizar el grafo del primer nivel, producto de las adiciones y eliminaciones de objetos, es $O(n^2)$. Por lo tanto, el número de operaciones realizadas en el paso 3 $T_3(n)$ es $O(n^2)$.

El número de operaciones realizadas en los pasos 4-18 depende de dos factores: la cantidad de operaciones realizadas en los pasos 5-17 para cada nivel de la jerarquía y el número de niveles que se construyen. Como se explicó en la sección 4.1.5, en el peor caso, el número de niveles construidos es $t = \log_2 n$ y el número de objetos de cada nivel es $n_i = \frac{n_{i-1}}{2}, i = 2..t$. Por otra parte, en las operaciones que se realizan en los pasos 5-17, los pasos más costosos son el 10, el 16 y el 17. A partir del análisis presentado en la sección 3.3.2, se tiene que el número de operaciones realizadas en el paso 10 es $O(n_i^2)$. De forma similar, el número de operaciones que se realiza en el paso 16 es $O(n_i \cdot n)$ en el peor caso y el número de operaciones que se realiza en el paso 17 es $O(n_i^2)$. A partir del análisis de la complejidad del algoritmo HClustR, se puede concluir que el número de operaciones realizadas en los pasos 4-18 es $O(n^2)$.

Finalmente, el número de operaciones realizadas en el Algoritmo 4.2 es $T_t(n) = T_3(n) + T_{4-17}(n)$. Por la regla de la suma, se tiene que $T_t(n)$ es $O(T_3(n), T_{4-17}(n))$ y por lo tanto, $T_t(n)$ es $O(n_2)$. Luego, la complejidad computacional del algoritmo DHClustR, en cualquiera de sus dos versiones, es $O(n^2)$.

4.3. Resultados experimentales

En esta sección, se presentan los resultados de un conjunto de experimentos en los que se evalúa a los algoritmos DHClustR-v1 y DHClustR-v2 de acuerdo a varios criterios. Los experimentos fueron realizados sobre las colecciones descritas en la sección 3.4.1.

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 DHClustR-v1 y DHClustR-v2, 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 propuestos en el procesamiento de las colecciones de prueba. Los aspectos evaluados en cada experimentos han sido los utilizados en varios trabajos para evaluar algoritmos jerárquicos (Gil-García, 2005; Gil-García *et al.*, 2005; Gil-García and Pons-Porrata, 2010). En los experimentos anteriores, se compararon los resultados obtenidos por DHClustR-

-v1 y DHClustR-v2 contra los resultados obtenidos por el algoritmo DHS (Gil-García and Pons-Porrata, 2010). 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.

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.

4.3.1. 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 comentadas en la sección 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 sección 3.4.2 se describieron las limitaciones de las medidas externas más utilizadas.

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 (Allan *et al.*, 2003) bajo el marco del proyecto TDT de la Universidad de Massachusetts (James, 2002). Bajo esta alternativa, para evaluar un algoritmo jerárquico 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 (Allan *et al.*, 2003). 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 (Larsen and Aone, 1999) (ver sección 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 (Gil-García and Pons-Porrata, 2010), 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.

Hasta este punto, se han descrito algunas variantes para la evaluación de algoritmos jerárquicos. De las alternativas descritas, la que ha sido más utilizada en la literatura es la alternativa 1 (Gil-García *et al.*, 2005; Gil-García and Pons-Porrata, 2008, 2010). Por tal motivo, se ha decidido utilizar la alternativa 1 para evaluar los algoritmos jerárquicos que se propongan en esta investigación.

4.3.2. Calidad de la jerarquía

En este experimento se comparan los algoritmos de acuerdo a la calidad de la jerarquía que forman para cada colección de documentos. Para determinar la calidad de la jerarquía se empleó la alternativa 1 que, como se explicó en la sección 4.3.1, es la más usada en la literatura para evaluar la calidad de los algoritmos jerárquicos. La medida externa utilizada con la alternativa 1 fue la medida FBcubed (Amigó *et al.*, 2009). Para una mejor comprensión de cómo se llevó a cabo este experimento, se describe a continuación el procedimiento que se siguió con la colección Reuter.

Inicialmente, todos los algoritmos se ejecutaron, sobre la colección Reuter, utilizando valores de β en el intervalo [0.05,0.35] y un incremento de 0.01; es decir, los valores de β que se utilizaron fueron β=0.05, β=0.06, β=0.07, ..., β=0.35. A continuación, se calculó el valor de FBcubed de la jerarquía formada por cada algoritmo, para cada valor de β usado. Para calcular el valor de FBcubed de una jerarquía, como se explicó en la sección 4.3.1, se calcula el valor de FBcubed de cada uno de los niveles y el mayor valor de FBcubed obtenido es el valor de FBcubed de dicha jerarquía. En la mayoría de los experimentos realizados, el nivel que alcanzó el mejor valor de FBcubed en cada jerarquía fue el tope de la jerarquía.

Posteriormente, se determinó la mejor jerarquía formada por cada algoritmo. Para el algoritmo DHS, se tomó como mejor jerarquía aquella que alcanza el mayor valor de FBcubed. Como los algoritmos DHClustR-v1 y

DHClustR-v2 son dependientes del orden de análisis de los objetos, se repitió veinte veces el experimento anterior, variando el orden de los documentos de la colección. Para estos dos algoritmos, se tomó como mejor jerarquía aquella que obtiene el mayor valor promedio de FBcubed. El valor de FBcubed obtenido por la mejor jerarquía de cada algoritmo, determina la mejor evaluación de dicho algoritmo sobre la colección Reuter. Para determinar la mejor evaluación de cada algoritmo con el resto de las colecciones, se utilizó un procedimiento similar al empleado con la colección Reuter.

Es importante mencionar que, a pesar de que los algoritmos DHClustR--v1 y DHClustR-v2 dependen del orden en que se analizan los documentos, la desviación standard del valor de FBcubed de la mejor jerarquía formada por estos algoritmos, para los diferentes órdenes, fue menor a 0.01, para cada valor de β. Esto significa que dicho valor de FBcubed varió poco para los diferentes órdenes; por lo tanto, se puede utilizar el valor promedio como el mejor desempeño de dichos algoritmos.

Durante el procesamiento de las colecciones se observó que, para valores de β mayores que 0.35 y menores a 0.05, el valor de calidad de los algoritmos evaluados decreció; por esta razón, no se emplearon en los experimentos valores de β fuera de este intervalo. En la Tabla 4.1 se muestra la mejor evaluación que cada algoritmo logra, según la alternativa 1 y la medida FBcubed, sobre las colecciones de prueba. En la Tabla 4.2 se muestran los valores de β para los cuales los algoritmos obtienen su mejor evaluación.

Tabla 4.1: Mejores evaluaciones de cada algoritmo en las colecciones de prueba. En cada colección, los valores más altos de calidad aparecen 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-v1	0.77	0.51	0.43	0.44	0.48	0.33
DHClustR-v2	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-v1	0.48	0.52	**0.51**	0.50	0.50	0.32
DHClustR-v2	**0.49**	**0.53**	**0.51**	**0.52**	**0.51**	**0.34**

Como puede verse en la Tabla 4.1, los algoritmos DHClustR-v1 y DHClustR-v2 obtienen en la mayoría de las colecciones, mejores valores

Tabla 4.2: Valores de β para los cuales cada algoritmo alcanza sus mejores evaluaciones sobre cada colección

	Colecciones					
Algoritmos	AFP	Reu-Te	Reu-Tr	Reuter	TDT	cacm
DHS	0.16	0.12	0.10	0.10	0.18	0.15
DHClustR-v1	0.21	0.20	0.22	0.23	0.33	0.26
DHClustR-v2	0.24	0.20	0.21	0.22	0.33	0.26
	Colecciones					
Algoritmos	TDT-1	TDT-2	TDT-3	TDT-4	TDT-5	cisi
DHS	0.17	0.19	0.17	0.17	0.17	0.09
DHClustR-v1	0.34	0.32	0.32	0.32	0.32	0.21
DHClustR-v2	0.34	0.32	0.33	0.33	0.33	0.21

de FBcubed que los que obtiene el algoritmo DHS. Más aún, el algoritmo DHClustR-v2 obtiene en casi todas las colecciones el mejor valor de calidad.

Para analizar los resultados de calidad presentados en la Tabla 4.1, se utilizó la metodología de evaluación de calidad de los agrupamientos, descrita en la sección 3.4.3. Esta metodología consiste en comparar dos a dos los algoritmos de agrupamiento, de acuerdo a los valores de calidad que obtienen sobre todas las colecciones de prueba. En la comparación entre un par de algoritmos A1 y A2, se determina el número de colecciones en las que cada algoritmo mejora/iguala al otro algoritmo y además, se determina la significancia estadística de los resultados. Como resultado de este análisis, en la Tabla 4.3 se muestra la *Matriz de dominancia* y la *Matriz de significancia estadística* de los valores de calidad obtenidos por cada algoritmo, sobre las colecciones de prueba.

Tabla 4.3: Matriz de dominancia y de significancia estadística de los valores de calidad obtenidos por cada algoritmo sobre las colecciones de prueba

	Matriz de dominancia		
Algoritmos	DHS	DHClustR-v1	DHClustR-v2
DHS	-	2/0	1/0
DHClustR-v1	10/0	-	0/2
DHClustR-v2	11/0	10/2	-
	Matriz de significancia estadística		
Algoritmos	DHS	DHClustR-v1	DHClustR-v2
DHS	-	<	<<
DHClustR-v1	>	-	<
DHClustR-v2	>>	>	-

Como se puede observar en la Tabla 4.3, los algoritmos DHClustR-v1 y DHClustR-v2 superan en casi todas las colecciones al algoritmo DHS. Adicionalmente, esta tabla muestra que los valores de calidad que obtiene DHClustR-v2 son significativamente superiores a los obtenidos por el algoritmo DHS.

Adicionalmente, como otra forma de evaluar la calidad de cada algoritmo, en la Tabla 4.4 se muestra el promedio de FBcubed que alcanzan todos los niveles de la mejor jerarquía formada por cada algoritmo y la desviación estándar del valor de FBcubed de dichos niveles. Note que, en caso de que se fuera a utilizar la estructura jerárquica completa y no solo el mejor nivel de ésta, sería importante que todos los niveles de la jerarquía alcanzaran un valor de FBcubed alto y relativamente cercano al del mejor nivel

Tabla 4.4: Promedio y Desviación estándar del valor de FBcubed de los niveles de la mejor jerarquía formada por cada algoritmo en las colecciones de prueba. Los promedios más altos aparecen en negrita

	Algoritmos					
	DHS		DHClustR-v1		DHClustR-v2	
Colección	prom.	desv.	prom.	desv.	prom.	desv.
AFP	0.52	0.24	**0.77**	2.97E-04	**0.77**	2.90E-03
Reu-Te	0.26	0.17	0.51	5.66E-05	**0.52**	0.01
Reu-Tr	0.21	0.15	**0.43**	7.07E-06	**0.43**	0.03
Reuter	0.24	0.16	0.43	5.54E-03	**0.44**	0.01
TDT	0.31	0.17	**0.48**	8.41E-04	**0.48**	0.01
TDT-1	0.36	0.15	0.52	5.68E-04	**0.53**	0.01
TDT-2	0.34	0.15	**0.50**	7.14E-04	**0.50**	3.48E-03
TDT-3	0.35	0.17	**0.50**	2.11E-03	**0.50**	0.01
TDT-4	0.34	0.17	0.50	9.56E-04	**0.51**	0.01
TDT-5	0.31	0.17	**0.48**	4.67E-04	**0.48**	0.01
cacm	0.25	0.04	**0.33**	2.55E-04	**0.33**	0.01
cisi	0.17	0.09	0.32	2.76E-04	**0.33**	0.01

Como se puede observar en la Tabla 4.4, los algoritmos DHClustR-v1 y DHClustR-v2 obtienen los promedios de FBcubed más altos en cada una de las colecciones usadas. Adicionalmente, el algoritmo DHClustR-v1 obtiene los valores de desviación estándar más bajos, seguido del algoritmo DHClustR-v2 que es el segundo mejor. Lo anterior muestra que el algoritmo DHS forma jerarquías en las cuales hay niveles con valores de FBcubed muy bajos; es decir, niveles con grupos de baja calidad, de acuerdo a la medida FBcubed.

4.3.3. Número de niveles y cantidad de grupos

En este experimento, se compara el número de niveles y la cantidad de grupos que contiene la mejor jerarquía formada por cada algoritmo para las colecciones de prueba. La Tabla 4.5 muestra, para cada colección de prueba, el número de niveles que tienen las mejores jerarquías formadas por cada algoritmo y cuantos grupos contienen dichas jerarquías.

Tabla 4.5: Número de niveles y cantidad de grupos de las mejores jerarquías formadas por cada algoritmo en las colecciones de prueba

	Algoritmos					
	DHS		DHClustR-v1		DHClustR-v2	
Colección	#niv.	#grup.	#niv.	#grup.	#niv.	#grup.
AFP	5	502	2	103	2	134
Reu-Te	7	2525	2	200	2	190
Reu-Tr	8	5408	2	329	3	378
Reuter	9	7680	2	524	3	564
TDT	8	11644	2	1751	3	2576
TDT-1	8	6383	3	1993	3	2007
TDT-2	8	5819	2	969	3	1448
TDT-3	8	7345	2	1154	3	1911
TDT-4	8	7463	3	1655	3	1896
TDT-5	8	8535	2	1187	3	2028
cacm	5	359	2	203	2	195
cisi	6	801	2	102	3	142

Como se puede observar en la Tabla 4.5, los algoritmos DHClustR-v1 y DHClustR-v2 construyen jerarquías que tienen menos niveles y menos grupos que las que construye el algoritmo DHS. Estos dos parámetros son importantes para aplicaciones como la recuperación y organización de información sobre la WWW (Beil *et al.*, 2002; Hammouda and Kamel, 2004; Fung *et al.*, 2003). En este tipo de aplicaciones, usualmente los resultados se organizan de forma jerárquica y el usuario puede navegar por esta jerarquía para localizar los resultados que son de su interés.

El número de niveles y grupos que tiene la jerarquía, depende del conjunto de objetos procesados. No obstante, en aplicaciones como las anteriormente comentadas, construir jerarquías con muchos niveles y muchos grupos le dificulta al usuario la navegación por dicha estructura y por lo tanto, la búsqueda de grupos de su interés (Fung *et al.*, 2003).

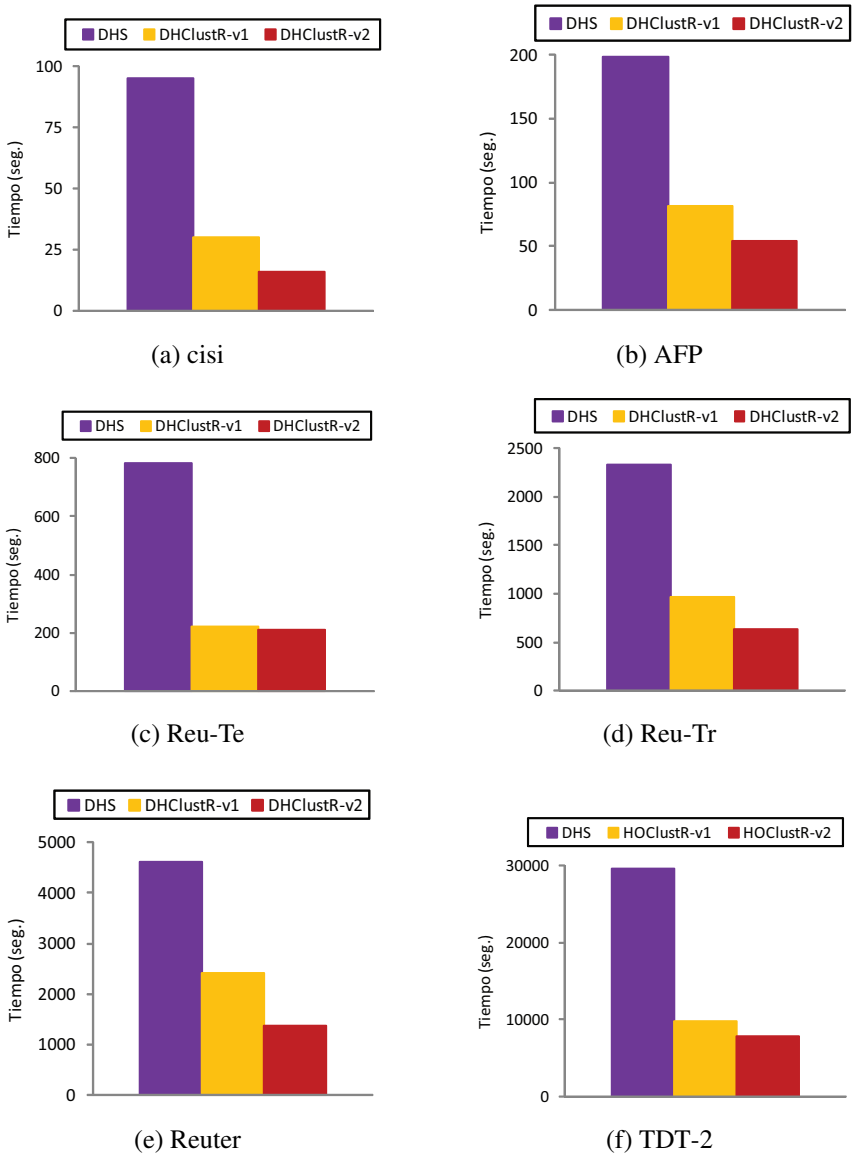
En los experimentos presentados en la sección 4.3.2 se observó que, para un mismo valor de β, el algoritmo DHS siempre construyó jerarquías con más niveles y más grupos que las construidas por los algoritmos DHClustR-v1 y DHClustR-v2. Adicionalmente, estos experimentos mostraron que el algoritmo DHClustR-v1 es el que construye las jerarquías de menos niveles y de menos grupos, en varios casos empatado con el algoritmo DHClustR-v2. No obstante, las diferencias entre los algoritmos DHClustR-v1 y DHClustR-v2 en cuanto al número de niveles y de grupos no es significativa.

4.3.4. Eficiencia

Como se demostró en la sección 4.2.2, los algoritmos DHClustR-v1 y DHClustR-v2 tienen una complejidad computacional de $O(n^2)$. Por lo tanto, dado que el algoritmo DHS tiene una complejidad de $O(n^3)$ (Gil-García and Pons-Porrata, 2010), tanto DHClustR-v1 como DHClustR-v2 serán capaces de procesar una colección en menos tiempo, que lo que le tomaría al algoritmo DHS procesar la misma colección. No obstante, se realizó un experimento en el cual se midió el tiempo que cada algoritmo emplea para agrupar jerárquicamente las colecciones de prueba, utilizando varios valores del umbral β. En la Figura 4.1, se muestra el tiempo que emplea cada algoritmo para agrupar algunas de las colecciones de prueba, utilizando cada algoritmo β=0.30.

Como se puede observar en la Figura 4.1, el algoritmo DHS tarda más tiempo en procesar las colecciones de prueba, que los algoritmos DHClustR-v1 y DHClustR-v2. Esta diferencia se hace más notable por el hecho de que, como se mencionó en la sección 4.3.3, los algoritmos DHClustR-v1 y DHClustR-v2 construyen jerarquías que tienen menos niveles y menos grupos que las construidas por DHS. En el resto de las colecciones y con otros valores de β se observó un comportamiento similar.

Como se mencionó anteriormente, los algoritmos DHClustR-v1 y DHClustR-v2 utilizan la misma medida para calcular la semejanza entre los objetos del primer nivel de la jerarquía; por lo tanto, el primer nivel de las jerarquías formadas por ambos algoritmos es el mismo. En los niveles siguientes, como cada algoritmo utiliza una estrategia diferente para calcular la semejanza entre los objetos, los grafos que se obtienen son diferentes y consecuentemente, los grupos obtenidos también pueden ser diferentes.



(a) cisi (b) AFP

(c) Reu-Te (d) Reu-Tr

(e) Reuter (f) TDT-2

Figura 4.1: Tiempo que emplea cada algoritmo para procesar las colecciones para β=0.30.

Para evaluar el impacto que tiene la estrategia propuesta para el cálculo de la semejanza en los niveles superiores, se ejecutaron los algoritmos DHClustR-v1 y DHClustR-v2 con la colección Reuter, usando diferentes valores de β, y se midió el tiempo que cada algoritmo tarda en construir el grafo del segundo nivel. Este experimento se repitió diez veces variando el orden de los documentos de la colección Reuter. En la Tabla 4.6, se muestra el tiempo promedio que emplea cada algoritmo en la construcción del grafo del segundo nivel, para cada valor de β usado con la colección Reuter.

Tabla 4.6: Tiempo promedio en segundos que cada algoritmo emplea, durante el procesamiento de la colección Reuter, en la construcción del segundo nivel

	Umbrales					
Colección	0.20	0.21	0.22	0.23	0.24	0.25
DHClustR-v1	38.92	41.57	40.41	46.87	54.61	54.36
DHClustR-v2	0.66	0.74	0.85	1.00	1.16	1.33

Como se puede observar en la Tabla 4.6, la estrategia propuesta en DHClustR-v2 para calcular la semejanza entre los objetos de los niveles superiores, es más rápida que la empleada por DHClustR-v1. Es importante recordar, que esta estrategia permite también obtener jerarquías con altos valores de calidad (ver sección 4.3.2) y con un número de niveles y grupos relativamente pequeño (ver sección 4.3.3).

4.4. Síntesis y conclusiones

En este capítulo se propusieron dos algoritmos de agrupamiento para la formación de jerarquías traslapdas: HClustR y DHClustR.

HClustR es un algoritmo jerárquico aglomerativo y estático, que construye una jerarquía de grupos traslapados a través de la aplicación sucesiva del algoritmo OClustR. Para construir el nivel inferior, se aplica el algoritmo OClustR sobre la colección de objetos. A partir de este punto, los objetos a agrupar en cada nivel de la jerarquía son los grupos construidos en el nivel anterior. El proceso de construcción de la jerarquía se detiene cuando en el grafo de β-semejanza pesado $\widetilde{G}_\beta$, que representa a la colección de objetos del nivel, todos los vértices son aislados.

Hay dos aspectos que determinan el funcionamiento del algoritmo HClustR: la forma en que se representan los objetos de los niveles superiores y la forma en que se determina la semejanza entre los objetos de los niveles superiores. De acuerdo a estos dos criterios, se presentaron dos variantes de HClustR. La primera variante, nombrada HClustR-v1, representa los objetos como vectores de números reales y utiliza la medida del coseno para calcular la semejanza entre los objetos del primer nivel. Para representar los objetos de los niveles superiores, esta variante utiliza el vector composición de los grupos del nivel inmediato inferior y, para calcular la semejanza entre dichos objetos, utiliza la medida *group-average*. La segunda variante, nombrada HClustR-v2, introduce una nueva forma de representar los objetos de los niveles superiores que es menos costosa que la utilizada por HClustR-v1. Adicionalmente, HClustR-v2 introduce una estrategia nueva para determinar la semejanza entre los objetos de los niveles superiores.

DHClustR es un algoritmo jerárquico aglomerativo y dinámico, que se basa en los conceptos introducidos por HClustR para la formación de la jerarquía de grupos. Este algoritmo propone además una estrategia para la actualización eficiente de esta jerarquía, cuando la colección cambia producto de múltiples adiciones y/o eliminaciones de objetos. Se presentaron dos variantes de DHClustR: DHClustR-v1 y DHClustR-v2, las cuales se basan en HClustR-v1 y HClustR-v2 respectivamente. Es importante mencionar que la complejidad computacional de DHClustR es $O(n^2)$ y que la misma es inferior a la del algoritmo DHS (Gil-García and Pons-Porrata, 2010) que es $O(n^3)$; por lo tanto, el algoritmo propuesto será capaz de agrupar una colección en menos tiempo de lo que le tomaría a DHS agrupar la misma colección. DHS es el único algoritmo jerárquico reportado en estado del arte, que permite formar jerarquías traslapadas y que es capaz de actualizar dicha jerarquía luego de cambios en la colección.

Se realizaron un conjunto de experimentos en los que se comparó, usando varias colecciones estándares, el comportamiento de las dos variantes de DHClustR en relación con el algoritmo DHS. Los experimentos se enfocaron en comparar la calidad de las jerarquías formadas por los tres algoritmos, así como el número de niveles y grupos de dichas jerarquías. A partir de estos experimentos se puede concluir que el algoritmo DHClustR-v2 obtiene los mejores valores de calidad y que dichos valores son significativamente superiores a los obtenidos por el algoritmo DHS. Estos experimentos mostraron también que, tanto DHClustR-v1 como DHClustR-v2, forman jerarquías

con menos niveles y menos grupos que el algoritmo DHS.

Adicionalmente, se realizaron otros dos experimentos en los cuales se evaluó: (i) el tiempo que emplea cada algoritmo en procesar las colecciones de prueba y (ii) el impacto que tiene en la eficiencia de DHClustR-v2 el uso de la nueva estrategia propuesta para el cálculo de la semejanza entre los objetos de los niveles superiores. A partir de estos experimentos, se evidenció que el algoritmo DHS procesa las colecciones de prueba en aproximadamente el doble del tiempo que los algoritmos DHClustR-v1 y DHClustR-v2 tardan en procesar las mismas colecciones. Adicionalmente, estos experimentos mostraron que la estrategia propuesta para el cálculo de la semejanza en los niveles superiores, le permite a DHClustR-v2 ser más eficiente que DHClustR-v1.

De todo el análisis anterior, se puede concluir que DHClustR-v2 representa una mejor opción, para enfrentar el problema del agrupamiento traslapado en un contexto jerárquico y dinámico, que los algoritmos reportados previamente. Con los resultados obtenidos en este capítulo, se consideran como cumplidos los objetivos particulares 3 y 4 de esta tesis doctoral.

Capítulo 5

Conclusiones

5.1. Conclusiones

El desarrollo de algoritmos de agrupamiento 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.

Los algoritmos traslapados que se han propuesto hasta el momento tienen un conjunto de limitaciones que pueden reducir su utilidad en ciertos problemas prácticos. Estas limitaciones están relacionadas principalmente con: (a) la necesidad de ajustar varios parámetros cuyos valores dependen de la colección a agrupar, (b) la construcción de un gran número de grupos, generalmente con un bajo promedio de elementos por grupo y (c) la obtención de agrupamientos con un alto nivel de traslape. Adicionalmente, la mayoría de los algoritmos traslapados existentes son incapaces de satisfacer nuevos requerimientos tales como: (i) la necesidad de actualizar el agrupamiento previamente construido, cuando cambia la colección y (ii) la necesidad de crear estructuras jerárquicas, en las cuales sea permitido el traslape entre los grupos de un mismo nivel.

En este trabajo se propusieron dos nuevos algoritmos de agrupamiento traslapado, DClustR y DHClustR, que abordan los requerimientos anterior-

mente comentados y que además, solucionan las limitaciones a), b) y c).

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 el tiempo que tarda cada algoritmo para formar la jerarquía de grupos.

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 traslapadas 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. Con los resultados presentados en los capítulos 3 y 4 se consideran cumplidos todos los objetivos particulares y, consecuentemente, el objetivo general de esta tesis doctoral.

5.2. Aportaciones

Las aportaciones de esta investigación son las siguientes:

a) Un algoritmo de agrupamiento llamado OClustR, que es estático y permite obtener un conjunto de grupos traslapado. OClustR forma agrupamientos con una calidad significativamente superior a la que tienen los agrupamientos formados por los algoritmos del estado del arte.

b) Un algoritmo de agrupamiento llamado DClustR, que es dinámico y permite obtener un conjunto de grupos traslapado. DClustR forma agrupamientos con una calidad significativamente superior a la que tienen los agrupamientos formados por los algoritmos del estado del arte y, adicionalmente, es más rápido en el procesamiento de múltiples adiciones y/o eliminaciones de objetos, que estos algoritmos.

c) Un algoritmo jerárquico aglomerativo y dinámico que permite obtener una jerarquía traslapada. DHClustR forma jerarquías traslapadas con una calidad significativamente superior a la que tienen las jerarquías formados por DHS; este algoritmo es el único algoritmo jerárquico y dinámico que forma jerarquías traslapadas. Adicionalmente, DHClustR es más rápido en el procesamiento de múltiples adiciones y/o eliminaciones de objetos, que DHS.

5.3. Trabajo futuro

Con base en los resultados alcanzados hasta el momento en esta investigación, se propone como trabajo futuro:

a) Evaluar el uso de la relevancia de los vértices como criterio para la eliminación de ws-grafos poco útiles en los algoritmos OCLustR y DClustR. La idea consistiría en ordenar los vértices en forma ascendente de acuerdo a su relevancia y, en ese orden, eliminar aquellos que sean prescindibles. Esta propuesta conllevaría la modificación del algoritmo DClustR, específicamente los pasos 5-18 del procedimiento *UpdCovCompt*.

b) Evaluar otras alternativas que permitan definir cuándo un ws-grafo definido por un vértice v tiene "casi todos" sus satélites cubiertos por otros vértices; *i.e.*, cuándo v es prescindible. La estrategia propuesta en la sección 3.2.2, compara los satélites que el ws-grafo formado por v comparte con otros ws-grafos con los satélites que solo v cubre y luego, con base en este análisis, decide si v es prescindible o no. No obstante, sería interesante estudiar la relación entre satélites compartidos y no compartidos.

c) Desarrollar versiones paralelas de los algoritmos propuestos en esta investigación.

5.4. Publicaciones

Los artículos derivados de esta investigación, hasta el momento, son los siguientes:

1. A. Pérez-Suárez, *et al*. “A New Incremental Algorithm for Overlapped Clustering” In: E. Bayro-Corrochano and J.-O. Eklundh (Eds.): CIARP 2009, LNCS 5856, pp. 497-504. Springer, Heidelberg (2009).
2. A Pérez-Suárez, *et al.* “A Dynamic Clustering Algorithm for Building Overlapped Clusters” por aparecer en el Journal Intelligent Data Analysis (JCR), 16(2), 2012.
3. A Pérez-Suárez, *et al.* “OClustR: A New Graph-based Algorithm for Overlapping Clustering” sometido al Journal Knowledge and Information Systems (JCR).
4. A Pérez-Suárez, *et al*. “A New Dynamic Clustering Algorithm for Overlapping Clustering” sometido al Journal Data Mining and Knowledge Discovery (JCR).

Bibliografía

Abella-Pérez R, Medina-Pagola J (2010) An incremental text segmentation by clustering cohesion. In: Proceedings of the International Workshop on Handling Concept Drift in Adaptive Information Systems: Importance, Challenges and Solutions (HaCDAIS 2010), pp 65–72

Aho A, Hopcroft J, Ullman J (1983) Data Structures and Algorithms. Addison-Wesley Publishing Company

Allan J, Feng A, Bolivar A (2003) Flexible intrinsic evaluation of hierarchical clustering for tdt. In: Proceedings of the Twelfth ACM International Conference on Information and Knowledge Management (CIKM 2003), pp 263–270

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

Aslam J, Pelekhov K, Rus D (1998) Static and dynamic information organization with star clusters. In: Proceedings of the seventh international conference on Information and knowledge management, pp 208–217

Aslam J, Pelekhov K, Rus D (2000) Using star clusters for filtering. In: Proceedings of the Ninth International Conference on Information and Knowledge Management, USA, pp 306–313

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

Bae E, Bailey J, Dong G (2010) A clustering comparison measure using density profiles and its application to the discovery of alternate clusterings. Data Mining and Knowledge Discovery 21(3):427–471

Bagga A, Baldwin B (1998) Entity-based cross-document coreferencing using the vector space model. In: Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics (COLING-ACL'98), pp 79–85

Bakus J, Hussin M, Kamel M (2002) A som-based document clustering using phrases. In: Proceedings of the 9th International Conference on Neural Information Processing (ICONIP'02), p 2212–2216

Banerjee A, Krumpelman C, Basu S, Mooney R, Ghosh J (2005) Model-based overlapping clustering. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (KDD2005), pp 532–537

Beil F, Ester M, Xu X (2002) Frequent term-based text clustering. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, New York, NY, USA, KDD '02, pp 436–442, DOI http://doi.acm.org/10.1145/775047.775110, URL http://doi.acm.org/10.1145/775047.775110

Berry M (2004) Survey of Text Mining, Clustering, Classification and Retrieval. Springer-Verlag

Chang L, Duarte M, Sucar L, Morales E (2010) Object class recognition using sift and bayesian networks. In: Proceedings of the Mexican International Conference on Artificial Intelligence (MICAI 2010), LNAI 6438, pp 56–66

Chung S, McLeod D (2005) Dynamic pattern mining: An incremental data clustering approach. Journal on Data Semantics II 3360:85–122

Costa G, Manco G, Ortale R (2010) An incremental clustering scheme for data deduplication. Data Mining and Knowledge Discovery 20(1):152–187

Crescenzi P, Kann V (1997) Approximation on the web: A compendium of np optimization problems. In: Proceedings of the International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'97), pp 111–118

Dom B (2001) An information-theoretic external cluster-validity measure. IBM Research Report, RJ 10219

Fung B, Wang K, Esther M (2003) Hierarchical document clustering using frequent itemsets. In: Proceedings of the Third SIAM International Conference on Data Mining, pp 59–70

Gago-Alonso A, Pérez-Suárez A, Medina-Pagola J (2007) Acons: a new algorithm for clustering documents. In: Proceedings of the 12th Iberoamerican Congress on Pattern Recognition (CIARP2007), LNCS 4756, pp 664–673

Gil-García R (2005) Algoritmos de agrupamiento sobre grafos y su paralelización. PhD thesis, Departamento de Ingenieria y Ciencia de los Computadores. Universidad Jaume I. España

Gil-García R, Badía-Contelles J, Pons-Porrata A (2005) Dynamic hierarchical compact clustering algorithm. In: Proceedings of the X Iberoamerican Congress on Pattern Recognition (CIARP2005), LNCS 3773, Springer-Verlag Berlin Heidelberg, pp 302–310

Gil-García R, Pons-Porrata A (2008) 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, pp 187–194

Gil-García R, Pons-Porrata A (2010) Dynamic hierarchical algorithms for document clustering. Pattern Recognition Letters 31(6):469–477

Gil-García RJ, Badía-Contelles JM, Pons-Porrata A (2003) Extended star clustering algorithm. In: Proceedings of the 8th Iberoamerican Congress on Pattern Recognition (CIARP2003), LNCS 2905, pp 480–487

Greengrass E (2001) Information retrieval: A survey. Tech. Rep. TR-R52-008-001

Gupta G, Liu A, Ghosh J (2010) Automated hierarchical density shaving: A robust, automated clustering and visualization framework for large biological datasets. IEEE/ACM Transactions on Computational Biology and Bioinformatics 7(2):223–237

Gurrutxaga I, Arbelaitz O, Martín J, Muguerza J, Pérez J, Perona I (2009) Sihc: A stable incremental hierarchical clustering algorithm. In: Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS2009), pp 300–304

Gusfield D (1997) Algorithms on strings, trees and sequences: computer science and computational biology, Cambridge University Press, chap 6

Halkidi M, Batistakis Y, Vazirgiannis M (2001) On clustering validation techniques. Journal of Intelligent Information Systems 17(2-3):107–145

Hammouda K, Kamel M (2004) Efficient phrase-based document indexing for web document clustering. IEEE Transactions on Knowledge and Data Engineering 16(10):1279–1296

Jain A, Dubes R (1988) Algorithms for clustering data. Prentice-Hall, Inc., Upper Saddle River, NJ, USA

Jain A, Murty M, Flynn P (1999) Data clustering: A review. ACM Computing Surveys 31(3):264–323

James A (ed) (2002) Topic detection and tracking: event-based information organization. Kluwer Academic Publishers, Norwell, MA, USA

Jiang T, Tuzhilin A (2009) Dynamic micro-targeting: fitness-based approach to predicting individual preferences. Knowledge and Information Systems 19(3):337–360

Jo T, Lee M (2007) The evaluation measure of text clustering for the variable number of clusters. In: Proceedings of the 4th international symposium on Neural Networks: Part II–Advances in Neural Networks, pp 871–879

Kianmehr K, Alshalalfa M, Alhajj R (2010) Fuzzy clustering-based discretization for gene expression classification. Knowledge and Information Systems 24(3):441–465

Knuth DE (1973) The Art of Computer Programming, vol 3. Addison-Wesley

Larsen B, Aone C (1999) Fast and effective text mining using linear-time document clustering. Knowledge Discovery and Data Mining pp 16–22

Liu Y, Liao X, Li X, Wu Z (2004) A tabu clustering algorithm for intrusion detection. Intelligent Data Analysis 8(4):325–344

Mahata P (2010) Exploratory consensus of hierarchical clusterings for melanoma and breast cancer. IEEE/ACM Transactions on Computational Biology and Bioinformatics 7(1):138–152

Martínez-Trinidad J, Guzmán-Arenas A (2001) The logical combinatorial approach to pattern recognition, an overview through selected works. Pattern Recognition 34(4):741–751

Meila M (2003) Comparing clusterings by the variation of information. In: Proceedings of the 16th Annual Conference on Learning Theory and 7th Kernel Workshop (COLT/Kernel 2003), LNAI 2777, pp 173–187

Narasimhamurthy A, Greene D, Hurley N, Cunningham P (2010) Partitioning large networks without breaking communities. Knowledge and Information Systems 25(2):345–369

Omran M, Engelbrecht A, Salman A (2007) An overview of clustering methods. Intelligent Data Analysis 11(6):583–605

Palshikar GK, Apte MM (2008) Collusion set detection using graph clustering. Data Mining and Knowledge Discovery 16(2):135–164

Pantel P, Lin D (2002) Efficiently clustering documents with committees. In: Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2002), pp 18–22

Pérez-Suárez A, Medina-Pagola J (2007) 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, pp 248–262

Pérez-Suárez A, Martínez-Trinidad J, Carrasco-Ochoa J, Medina-Pagola J (2009) A new incremental algorithm for overlapped clustering. In: Proceedings of the 14th Iberoamerican Congress on Pattern Recognition (CIARP2009), LNCS 5856, pp 497–504

Pfitzner D, Leibbrandt R, Powers D (2009) Characterization and evaluation of similarity measures for pairs of clusterings. Knowledge and Information Systems 19(3):361–394

Pons-Porrata A, Ruiz-Shulcloper J, Berlanga-Llavorí R, Santiesteban-Alganza Y (2002) Un algoritmo incremental para la obtención de cubrimientos con datos mezclados. Reconocimiento de Patrones Avances y Perspectivas Research on Computing Science, CIARP'2002 pp 405–416

Pons-Porrata A, Berlanga-Llavorí R, Ruiz-Shulcloper J, Pérez-Martínez JM (2004) Jerartop: A new topic detection system. In Proceedings of the 9th Iberoamerican Congress on Pattern Recognition, LNCS 3287, Springer Verlag pp 446–453

Puzicha J, Hofmann T, Buhmann J (2000) A theory of proximity based clustering: Structure detection by optimization. PATREC: Pattern Recognition 33(4):617–634

Rosenberg A, Hirschberg J (2007) V-measure: A conditional entropy-based external cluster evaluation measure. In: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pp 410–420

Roth V, Braun ML, Lange T, Buhmann JM (2002) Stability-based model order selection in clustering with applications to gene expression data. In: Proceedings of the International Conference on Artificial Neural Networks, pp 607–612

Salton G, Wong A, Yang CS (1975) A vector space model for automatic indexing. Commun ACM 18(11):613–620, DOI http://doi.acm.org/10.1145/361219.361220

Schmidt J, Hapfelmeier A, Mueller M, Perneczky R, Kurz A, Drzezga A, Kramer S (2010) Interpreting pet scans by structured patient data: a data mining case study in dementia research. Knowledge and Information Systems 24(1):149–170

Sibson R (1973) An optimally efficient algorithm for the single link cluster method. Computer Journal 16:30–34

Stein B, Meyer S, Wissbrock F (2003) On cluster validity and the information need of users. In: Proceedings of the 3rd IASTED International Conference on Artificial Intelligence and Applications (AIA'03), pp 216–221

Steinbach M, Karypis G, Kumar V (2000) A comparison of document clustering techniques. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2000)

Strehl A (2002) Relationship-based clustering and cluster ensembles for high-dimensional data mining. PhD thesis, The University of Texas at Austin

Wai-chiu W, Fu A (2000) Incremental document clustering for web page classification. In: IEEE 2000 International Conference on Information Society in the 21st Century: Emerging technologies and new challenges

Widyantoro D, Ioerger T, Yen J (2002) An incremental approach to building a cluster hierarchy. In: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM2002), pp 705–708

Xu W, Liu X, Gong Y (2003) Document clustering based on non-negative matrix factorization. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'03), pp 267–273

Yeung KY, Haynor DR, Ruzzo WL (2000) Validating clustering for gene expression data. Bioinformatics 17:309–318

Zamir O, Etziony O (1998) Web document clustering: A feasibility demonstration. In: Proceedings of the 21st Annual International ACM SIGIR Conference, pp 46–54

Zhao Y, Karypis G (2001) Criterion functions for document clustering: Experiments and analysis. Tech. Rep. 01–40, Department of Computer Science, University of Minnesota, Minneapolis, MN

Zhao Y, Karypis G (2002) Evaluation of hierarchical clustering algorithms for document datasets. In: Proceedings of the eleventh international conference on Information and knowledge management, ACM, New York, NY, USA, CIKM '02, pp 515–524, DOI http://doi.acm.org/10.1145/584792.584877, URL http://doi.acm.org/10.1145/584792.584877

Zhao Y, Karypis G, Fayyad U (2005) Hierarchical clustering algorithms for document datasets. Data Mining and Knowledge Discovery 10(2):141–168

Apéndice A

Notaciones

β	Umbral mínimo de semejanza
$S(o_i, o_j)$	Valor de semejanza entre los objetos o_i y o_j
$A \subseteq B$	A es subconjunto de B (Si A y B son conjuntos)
G_β	Grafo de β-semejanza
E_β	Conjunto de aristas de G_β
$G_1 \subseteq G_2$	G_1 es sub-grafo de G_2 (Si G_1 y G_2 son grafos)
$G_{max-\beta}$	Grafo de máxima β-semejanza
$E_{max-\beta}$	Conjunto de aristas de $G_{max-\beta}$
$CDET(g_i, c_j)$	Costo de detección de un grupo g_j respecto a una clase c_i
$CDET(g_j)$	Costo de detección del grupo g_j
$CNAV(h_i)$	Costo de navegar desde la raíz de un árbol hasta el nodo h_i
$A(g_i)$	Conjunto de ancestros de un grupo g_i
H_{c_i}	Histograma de semejanza de un grupo c_i
HR_{c_i}	Cociente de H_{c_i}
HR_{min}	Umbral del algoritmo SHC
ϵ	Umbral del algoritmo SHC
$\widetilde{G}_\beta$	Grafo de β-semejanza pesado
$\widetilde{E}_\beta$	Conjunto de aristas de $\widetilde{G}_\beta$
$G^\star$	Sub-grafo pesado en forma de estrella
$V^\star$	Conjunto de vértices de $G^\star$
$E^\star$	Conjunto de aristas de $G^\star$

$Intra\_sim(G^\star)$	Semejanza intra-grupo de $G^\star$
$Aprox\_Intra\_sim(G^\star)$	Semejanza intra-grupo aproximada de $G^\star$
$O(\cdot)$	Notación asintótica para la complejidad de algo ritmos (*Big O*)
$SEM_{ave}(G_1, G_2)$	Semejanza entre los conjuntos G_1 y G_2 de acuer do a la medida group-average
$Core(G)$	Núcleo del grupo G
$S_B(G_1, G_2)$	Valor de semejanza *tipo-B* que existe entre los grupos G_1 y G_2
$S_C(G_1, G_2)$	Valor de semejanza *tipo-C* que existe entre los grupos G_1 y G_2
$S_T(G_1, G_2)$	Valor de semejanza que existe entre los grupos G_1 y G_2
$\in$	Pertenencia
$\notin$	No pertenencia
$\cup$	Unión de conjuntos
$\setminus$	Diferencia de conjuntos
$\emptyset$	Conjunto vacío
$\forall$	Cuantificador universal
$\exists$	Cuantificador existencial
$\wedge$	Operador lógico de conjunción
$\lvert \cdot \rvert$	Operador de cardinalidad

Apéndice B

Acrónimos

WWW	World Wide Web
OClustR	Object Clustering based on Relevance
DClustR	Dynamic Clustering based on Relevance
HClustR	Hierarchical Clustering based on Relevance
DHClustR	Dynamic Hierarchical Clustering based on Relevance
FOM	Figure of Merit
HClustR-v1	Variante 1 del algoritmo HClustR
HClustR-v2	Variante 2 del algoritmo HClustR
DHClustR-v1	Variante 1 del algoritmo DHClustR
DHClustR-v2	Variante 2 del algoritmo DHClustR
s-grafo	Sub-grafo en forma de estrella
CDET	Costo de Detección
VAL_MIN	Valor mínimo
STC	Suffix Tree Clustering
SHC	Similarity Histogram Clustering
ISC	Incremental Strong Compact
ICSD	Incremental Clustering Based on Strength Decision
DHS	Dynamic Hierarchical Star
Estar	Extended Star
Gstar	Generalized Star
VSM	Vector Space Model
ws-grafo	Sub-grafo pesado en forma de estrella

ACONS	Algorithm based on Condensed Star
TR	Traslape relativo
RAM	*Random Access Memory*
GHz	*Giga Hertz*