RNPS No. 2143
ISSN 2072-6260
Versión Digital

SERIE GRIS

REPORTE TÉCNICO
Minería de Datos

Algoritmos de agrupamiento para colecciones de documentos.

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

RT 005 **Noviembre 2008**

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

RNPS No. 2143
ISSN 2072-6260

SERIE GRIS

REPORTE TÉCNICO
Minería de Datos

Algoritmos de agrupamiento para colecciones de documentos

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

RT 005 **Noviembre 2008**

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

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

Algoritmos de agrupamiento para colecciones de documentos

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

[1] Departamento de Minería de Datos Centro de Aplicaciones de Tecnología de Avanzada (CENATAV), 7a ♯ 21812 e/ 218 y 222, Rpto. Siboney, Playa, C.P. 12200, La Habana, Cuba.

[2] Coordinación de Ciencias Computacionales, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Luis Enrique Erro No. 1, Sta. María Tonantzintla, Puebla, CP: 72840, Mexico.

{asuarez,ggarcia,jmedina}@cenatav.co.cu

{fmartine,ariel}@inaoep.mx

RT 005 CENATAV

Fecha del camera ready: Noviembre 2008

Resumen: El presente reporte técnico aborda la problemática del agrupamiento de documentos. En el mismo se realiza un análisis crítico de los métodos más importantes reportados en la literatura. Del conjunto de métodos analizados, se seleccionó un subconjunto que incluye a algoritmos de diferente tipo. Los algoritmos seleccionados fueron programados en C++ y se integraron a una plataforma desarrollada en .NET que permite la realización de experimentos con diferentes respositorios de documentos.

Palabras clave: Minería de datos, minería de texto, agrupamiento de documentos

Abstract: This technical report faces the document clustering problem. A critical study of the most relevant methods reported in the scientific literature is presented. From the set of studied algorithms, different kinds of algorithms were selected and programmed in C++. The selected algorithms were included in a framework developed in .NET which allows us to make experiments with different document datasets.

Keywords: Data mining, text mining, document clustering

Índice general

Lista de Algoritmos

1. Introducción

En la década de los noventa, disciplinas como la minería de datos, el aprendizaje automático, la minería de textos y el reconocimiento de patrones resultaron de gran ayuda para el hombre en su constante afán de extraer información relevante de un conjunto de datos. Hoy en día, el volumen de información almacenada crece a una velocidad verdaderamente increíble; luego, el análisis de ésta resulta cada vez más complejo. Es en este punto donde la aplicación de técnicas de las disciplinas mencionadas adquiere un papel importante.

Una de las técnicas que ha demostrado ser de utilidad en varios contextos es el agrupamiento o estructuración de datos. El agrupamiento es el proceso de organizar o estructurar una colección de objetos en clases o *clusters* de forma que los objetos contenidos en un mismo grupo sean más semejantes en relación con objetos que pertenezcan a grupos diferentes [26].

Algunas de las aplicaciones de los algoritmos de agrupamiento están asociadas con áreas como: (a) la biología, donde se han utilizado algoritmos de agrupamiento para el estudio de enfermedades como el cáncer de mama [44] o para el estudio de secuencias de genes [24],[78]; (b) en contextos como la detección y seguimiento de sucesos en flujos de noticias o correos [9],[59],[14]; (c) el procesamiento de imágenes [37],[8],[39]; (d) el procesamiento de datos espaciales [29],[83],[48], donde se busca descubrir estructuras así como analizar flujos en datos que son generados continuamente. En otros contextos, como el procesamiento de documentos web, se ha utilizado el agrupamiento como herramienta para el descubrimiento de patrones en el comportamiento de usuarios que navegan en distintos sitios web [54],[55],[50], en el descubrimiento de relaciones presentes entre los tópicos de diferentes sitios visitados [27] así como para personalizar, en dependencia de los gustos de ciertos clientes, los servicios brindados por un portal web [81].

En la literatura existen diversos algoritmos de agrupamiento reportados, observándose en todos un conjunto de pasos involucrados: (i) representación de los objetos (opcionalmente incluye extracción o selección de variables), (ii) definición de una medida de proximidad o semejanza entre los objetos, (iii) agrupar los objetos utilizando algún criterio de agrupamiento o heurística. Además de estos pasos, y aunque no forma parte como tal del proceso de agrupamiento, es común incluir un proceso de evaluación de los resultados obtenidos.

Este reporte centra su atención en los algoritmos de agrupamiento de documentos y el mismo está organizado como sigue: en la sección 2 se describen las características de los trabajos más importantes reportados en la literatura referente al agrupamiento de documentos. En la sección 3 se expone un conjunto de experimentos realizados en una selección de los algoritmos descritos en la sección anterior. Finalmente, la sección 4 expone las conclusiones de este reporte.

2. Algoritmos de agrupamiento

Los algoritmos de agrupamiento de documentos pueden clasificarse atendiendo a varios criterios [85],[33],[26]. Algunos de estos criterios son: (*i*) la forma en que procesan los objetos, (*ii*) atendiendo a cómo se considera la pertenencia de los objetos a los grupos formados, (*iii*) la forma en que se organizan o se relacionan los grupos obtenidos y (*iv*) el mecanismo en que se basan para agrupar los objetos.

En esta sección se describen los principales trabajos reportados en la literatura referentes al agrupamiento de documentos, mostrándose para cada uno sus características así como sus limitaciones.

2.1. Algoritmos de pasada simple

Los algoritmos de *pasada simple* son aquellos en los que cada vez que se tiene que agrupar un documento, éste se compara con cada uno de los grupos existentes y se adiciona al grupo que logró el máximo valor de semejanza. En caso de no existir semejanza con algún grupo el documento conforma un grupo independiente.

De forma general estos algoritmos definen una función de semejanza entre un grupo y un objeto. Utilizando esta función y un umbral especificado previamente, se decide a qué grupo asignar dicho objeto. Las funciones que se han usado con frecuencia son:

- *Promedio*. Dado el grupo G y el elemento O, la semejanza entre ellos se calcula como el promedio de las semejanzas existentes entre O y todos los elementos de C.
- *Máxima semejanza*. Dado el grupo C y el elemento O, la semejanza entre ellos se calcula como el máximo entre las semejanzas existentes entre O y todos los elementos de C.
- *Representante*. En esta función se asume que cada grupo está representado por un objeto que pertenece al grupo o se calcula a través de los elementos del grupo. Luego, la semejanza entre el grupo C y el elemento O, se calcula como la semejanza entre O y R_C, siendo este último el representante del grupo C.

Un ejemplo de este tipo de algoritmos es el *Single-Pass* que ha sido utilizado en sistemas para la detección de tópicos [11],[70]. Este algoritmo representa cada grupo a través de su representante, el cual se calcula como la media de los objetos pertenecientes al grupo. El algoritmo funciona de la siguiente forma: cada elemento se compara con los representantes de cada grupo existente para determinar su semejanza, seleccionándose el grupo más semejante al objeto mientras esta semejanza sea superior a un cierto umbral β predefinido. A continuación, el objeto es asignado al grupo determinado en el paso anterior y se ajusta consecuentemente el representante de dicho grupo. En caso de que no existan grupos con semejanzas superiores a β, se crea un nuevo grupo que contenga al objeto. La complejidad computacional de este algoritmo es lineal.

El pseudo-código del algoritmo Single-Pass puede observarse en el Algoritmo 1. En este caso, la versión del algoritmo Single-Pass que se muestra asume que la semejanza entre un objeto y un grupo es a través del representante del grupo y que éste se calcula como el promedio de los elementos del grupo.

Algoritmo 1: Single-Pass

Input: $D := \{d_1, d_2, \ldots, d_n\}$ - document collection,
β - similarity threshold
Output: SC - Set of clusters

```
1  SC := ∅;
2  forall d_j ∈ D do
3      i := −1;
4      i := arg_i máx{Sim(d_j, C_i.Rep), C_i ∈ SC};
5      if (i ≠ −1 and Sim(d_j, C_i.Rep) ≥ β) then
6          C_i := C_i ∪ {d_j};
7          C_i.Rep := Average(C_i); // Updating the cluster's representative
8      end
9      else
10         C := {d_j};
11         C.Rep := d_j;
12         SC := SC ∪ C;
13     end
14 end
```

El Single-Pass es un algoritmo incremental y los grupos que se obtienen a través del mismo son disjuntos. La principal limitación de este algoritmo es que los grupos formados pueden variar de acuerdo al orden en que se procesan los documentos. Esta variabilidad en el resultado del agrupamiento también está provocada por el criterio de asignación de elementos a grupos, ya que cuando existe más de un grupo con semejanza máxima al documento que se está procesando la selección del grupo al cual se asigne el documento determinará el agrupamiento final.

2.2. Algoritmos basados en optimización

Estos algoritmos producen una estructuración de la colección de documentos a través de la optimización de una función que estima la calidad del agrupamiento realizado hasta el momento. Teniendo en cuenta esta función, comúnmente llamada *función objetivo*, el algoritmo iterativamente procede a "re-asignar" documentos de un grupo a otro siempre y cuando se optimice el valor de dicha función.

La definición de esta función objetivo puede estar relacionada con medidas de semejanza o distancia entre los documentos que componen a cada uno de los grupos. Existen

varias funciones objetivos reportadas en la literatura; sin embargo, la más utilizada en este tipo de algoritmos es la que estima el error cuadrático [85].

Esta función puede ser definida utilizando una medida de distancia (1) ó de semejanza (2).

$$E = \sum_{i=1}^{k} \sum_{p \in C_i}^{|C_i|} |p - m_i|^2 \tag{1}$$

$$E = \sum_{i=1}^{k} \sum_{p \in C_i}^{|C_i|} F(p, m_i) \tag{2}$$

En las ecuaciones anteriores, p representa a los documentos que pertenecen a cada grupo $C_i, i = 1...k$ que tiene por representante a m_i y $F(p, m_i)$ representa la función que estima la semejanza entre p y m_i. En el caso de documentos, la función de semejanza que se utiliza con más frecuencia es la medida del coseno.

Un algoritmo de optimización que utiliza esta función y que además resulta el más simple y frecuentemente utilizado es el *k-means* [28],[53]. Este algoritmo representa los grupos a través de representantes o centroides que se calculan por la media (o promedio pesado) de los elementos pertenecientes al grupo. Inicialmente, el algoritmo selecciona aleatoriamente k elementos como centros iniciales de los grupos y procesa cada objeto restante asignándolo al grupo con el cual tenga la mínima distancia o mayor semejanza. A partir de este punto se recalculan los centroides y se vuelve a iterar por los objetos re-asignando cada uno a su grupo más cercano.

Este proceso se repite hasta que se alcance algún criterio de convergencia. Usualmente el criterio puede ser: (a) no se afecta el agrupamiento; es decir, ningún objeto cambia de grupo o (b) la función objetivo deja de crecer o decrecer (en dependencia si se utilizó la función 2 ó la 1). Los grupos que se obtienen producto de este algoritmo son grupos disjuntos. La complejidad computacional de este algoritmo es $O(nkT)$, donde T representa la cantidad de iteraciones hasta la convergencia del algoritmo.

El pseudo-código del algoritmo k-means puede observarse en el Algoritmo 2. En el pseudo-código presentado, se asumió que existe una función *Rand* que selecciona aleatoriamente un documento de un conjunto dado y una función *Evaluate* que verifica el cumplimiento de las condiciones de paro (a) y (b).

Este algoritmo tiene varias deficiencias importantes. En primer lugar sólo puede ser utilizado si los objetos están descritos en un espacio métrico, no agrupa correctamente los objetos aislados y puede caer en mínimos locales de la función objetivo. Otra deficiencia importante es que la calidad del agrupamiento resultante está determinada por la selección inicial de los centroides. Por último, es importante mencionar que en este algoritmo es necesario determinar *a priori* el número de grupos en los que se desea particionar la colección.

Algoritmo 2: K-means

Input: $D := \{d_1, d_2, \ldots, d_n\}$- document collection,
k - number of groups to be formed
Output: SC - Set of k clusters

```
1  SC := ∅;
   // Selecting initial k seeds
2  for i = 1 to k do
3      r := {d | d = Rand(D) ∧ d was not previously selected};
4      C_i := {r};
5      C_i.Center := r;
6      SC := SC ∪ C_i;
7  end
8  repeat
       // forming the set of clusters
9      forall d_j ∈ D do
10         i := arg_i min{F(d_j, C_i.Center), C_i ∈ SC};
11         C_i := C_i ∪ d_j;
12     end
13     cond := Evaluate(a,b) ;                    // Evaluating the stop-conditions
       // updating cluster's representative
14     forall C_i ∈ SC do
15         C_i.Center := Average(C_i);
16         C_i := ∅;
17     end
18 until cond ≠ true ;
```

2.3. Algoritmos jerárquicos aglomerativos

Este tipo de algoritmos comienza considerando como un grupo a cada documento de la colección y a partir de este punto comienza un proceso en el que iterativamente va mezclando los dos grupos más semejantes hasta obtener un solo grupo que contiene a todos los elementos. Adicionalmente a este criterio de convergencia y por motivos prácticos, puede ser de utilidad no trabajar con toda la jerarquía sino sólo con un nivel de ésta. Este nivel o partición de la jerarquía puede obtenerse aplicando otros criterios como por ejemplo: (a) la inclusión de un parámetro que estime la semejanza mínima que deben tener dos grupos para poder ser mezclados y (b) la definición de una cantidad de grupos a obtener.

El punto clave en estos algoritmos es la función o método que determine cuándo dos grupos son cercanos o semejantes. Varios trabajos se han desarrollado con este propósito [32],[36]. En dependencia de la función que se considere pueden obtenerse diferentes métodos; los más utilizados para agrupar documentos son el *Single-link*, *Complete-Link* y *Average-Link*.

El Single-link [67] (SLINK) se caracteriza por calcular la semejanza entre dos grupos C_i y C_j a través de la mayor semejanza existente entre un elemento $p \in C_i$ y uno $q \in C_j$.

Teniendo en cuenta esta definición de semejanza entre grupos, el algoritmo iterativamente une los grupos que tengan mayor semejanza.

El Complete-Link [79] (CLINK) calcula la semejanza entre dos grupos C_i y C_j a través de la menor semejanza existente entre un elemento $p \in C_i$ y uno $q \in C_j$. Teniendo en cuenta esta definición de semejanza entre grupos, el algoritmo iterativamente une los grupos que tengan mayor semejanza.

De forma general, estos algoritmos obtienen malos resultados debido a que utilizan muy poca información (Single-link) o a que asumen que todos los documentos en un grupo están fuertemente relacionados (Complete-link) [85]. Un algoritmo que soluciona estos problemas, al utilizar una estrategia que representa un punto medio entre las anteriores, es el Average-link o UPGMA, como también es conocido [32].

El Average-link calcula la semejanza entre un par de grupos como el promedio de semejanza existente entre los objetos que componen a dichos grupos. Utilizando esta función, el algoritmo Average-link logra obtener grupos más cohesionados y que a la vez son menos sensibles al ruido.

Las funciones de semejanza utilizadas por los algoritmos anteriores se exponen en la tabla 1.

Tabla 1. Determinación de la semejanza entre grupos

Algoritmo	**Fórmula de la función de semejanza**
SLINK	$\text{mín}_{d_i \in Ci, d_j \in C_j} \, d(d_i, d_j)$
CLINK	$\frac{1}{\lvert C_i \rvert \lvert C_j \rvert} \sum_{d_i \in Ci} \sum_{d_j \in C_j} d(d_i, d_j)$
UPGMA	$\text{máx}_{d_i \in Ci, d_j \in C_j} \, d(d_i, d_j)$

La complejidad computacional de estos algoritmos es de $O(n^3)$; sin embargo, se pueden lograr implementaciones con complejidad computacional de $O(n^2)$ [20]. La principal deficiencia de estos algoritmos es que los grupos formados dependen del orden de análisis de los elementos, además, en el caso del Single-link, los grupos formados presentan encadenamiento y como consecuencia son poco cohesionados. Por otra parte, los grupos formados por Complete-link y Average-link tienden a tener forma esférica.

Cómo los tres algoritmos comparten pasos dentro de su funcionamiento y sólo se diferencian en la función de semejanza (Sim), se muestra un sólo pseudo-código para los tres algoritmos (ver Algoritmo 3).

2.4. Algoritmos jerárquicos divisivos

Los algoritmos de este tipo asumen inicialmente que todos los documentos pertenecen a un mismo grupo. A partir de este punto y aplicando técnicas particionales se realiza un proceso en el que iterativamente se selecciona un grupo para ser dividido en dos. Este proceso termina cuando se satisface algún criterio de convergencia. Generalmente, los

Algoritmo 3: Aglomerativos

Input: $D := \{d_1, d_2, \ldots, d_n\}$- document collection
Output: SC- Set of clusters

```
1  SC := ∅;
2  forall d_j ∈ D do  SC := SC ∪ {d_j};
3  while stop-conditions (a) or (b) are not satisfied do
4      temp := 0;
5      k := 0;
6      m := 0;
7      forall pair of cluster C_i and C_j ∈ SC do
8          if Sim(C_i, C_j) > temp then
9              temp := Sim(C_i, C_j);
10             k := i;
11             m := j;
12         end
13     end
14     "Join clusters C_k and C_m";
15     "Update SC";
16 end
```

criterios utilizados pueden son: (a) se alcanza una cantidad de grupos determinada o (b) los grupos formados satisfacen algún criterio de cohesión.

Teniendo en cuenta lo anteriormente mencionado podría utilizarse cualquier algoritmo particional con el cual ir realizando divisiones sucesivas y de esta forma, lograr la jerarquía deseada. Algunos de los algoritmos particionales utilizados con estos propósitos pueden consultarse en [85].

El algoritmo divisivo más utilizado es el *Bisecting K-means* [73], que utiliza el algoritmo particional k-means para iterativamente ir dividiendo un grupo seleccionado. Generalmente, el criterio para seleccionar el grupo puede ser: (i) el grupo más grande, (ii) el de menor semejanza intra-grupo o (iii) una combinación de ambos. Adicionalmente, conociendo que el resultado del algoritmo K-means depende de la selección de las semillas iniciales, se puede ejecutar el proceso de división del grupo seleccionado un número de veces determinada. Luego, utilizando una función de aptitud se puede seleccionar la división que produce el mejor resultado.

El pseudo-código del algoritmo Bisecting K-means puede observarse en el Algoritmo 4. En este pseudo-código, se asumió la existencia de las funciones:

a) F, que asigna un valor de aptitud a cada grupo existente de acuerdo al criterio utilizado para seleccionar un grupo a dividir.
b) $K - means$, que invoca al algoritmo del mismo nombre.

La complejidad computacional de este algoritmo es de $O(n^2)$ y al utilizar K-means como algoritmo para dividir los grupos presenta las mismas deficiencias de éste (ver 2.2). Además

Algoritmo 4: Bisecting K-means

Input: $D := \{d_1, d_2, \ldots, d_n\}$ - document collection,
k - number of iterations

Output: SC - Set of clusters

```
1  SC := D;
2  while stop-conditions (a) or (b) are not satisfied do
3      C := arg max{F(C_i) | C_i ∈ SC};
4      "Delete C from SC";
5      temp := K-means(C,2);
6      i := 0;
7      h := 0;
8      temp_1 := ∅;
9      while i < k do
10         m := Evaluate(SC ∪ temp);
11         if m > h then
12             h := m;
13             temp_1 := temp;
14         end
15         i++;
16         temp := K-means(C,2);
17     end
18     SC := SC ∪ temp_1;
19 end
```

de lo anterior, existen trabajos que plantean que, respecto a la calidad del agrupamiento, estos algoritmos obtienen peores resultados que sus contrapartes aglomerativas [41].

2.5. Algoritmos basados en árboles

Este tipo de algoritmos, como su nombre lo indica, crea un árbol con los documentos de acuerdo a determinados criterios; de esta forma, los nodos del árbol resumen las características de los documentos que contienen. Ejemplos de este tipo de algoritmo son el STC [84] y el DC-tree [80].

El algoritmo STC (*Suffix Tree Clustering*) fue creado para agrupar los *snippets*[3] devueltos en una consulta web. Este algoritmo, a diferencia de muchos otros algoritmos de agrupamiento, representa a los documentos como una secuencia ordenada de palabras logrando de esta forma utilizar la información sintáctica que se puede extraer de esta secuencia para realizar el agrupamiento.

Este algoritmo está compuesto por tres pasos. El primer paso se encarga de realizar un proceso de lematización de los documentos (*snippets*) a agrupar y eliminar todo aquello que no sea una palabra. Al final de este paso cada documento está representado por una

[3] Los *snippets* son los pequeños textos utilizados por los sistemas de búsqueda como Google para describir los resultados de las búsquedas

secuencia de palabras que mantiene el orden de acuerdo al documento original. El segundo paso tiene como objetivo construir los *grupos base* que serán utilizados para obtener el agrupamiento final.

Estos grupos base se obtienen del árbol de sufijos [76],[25] que se puede construir a partir de todos los posibles sufijos que se pueden descubrir utilizando todos los documentos de la colección. Cada nodo de este árbol contiene el conjunto de documentos que tienen en común el sufijo formado por el camino desde la raíz del árbol hasta el nodo en cuestión. Cada conjunto de documentos perteneciente a un nodo formará un grupo base.

El último paso aplica una heurística similar a la del algoritmo Single-link en la que, utilizando cierta medida de semejanza entre grupos, se unen iterativamente los grupos base que sean semejantes. El agrupamiento resultante está formado por los grupos construidos en este tercer paso. Los grupos obtenidos pueden ser solapados.

El pseudo-código del algoritmo *STC* puede observarse en el Algoritmo 5. En este pseudo-código se asumió la existencia de las funciones:

a) *Pre-Processing*, que se encarga de representar los documentos que se desean agrupar.

b) *BuildSuffixTree*, que se encarga de construir el árbol de sufijos.

Algoritmo 5: STC

Input: $D := \{d_1, d_2, \ldots, d_n\}$- document collection
Output: SC- Set of clusters

```
1  SC := ∅;
2  D_p := Pre-Processing(D);
3  T := BuildSuffixTree(D_p);
   // Forming bases clusters
4  forall n ∈ T.Nodes do SC := SC ∪ n.Docs;
   // Combining base clusters
5  while stop-condition is not satisfied do
6      temp := 0;
7      k := 0;
8      m := 0;
9      forall pair of cluster C_i and C_j ∈ SC do
10         if Sim(C_i, C_j) > temp then
11             temp := Sim(C_i, C_j);
12             k := i;
13             m := j;
14         end
15     end
16     "Join clusters C_k and C_m";
17     "Update SC";
18 end
```

El algoritmo DC-Tree (*Document Clustering Tree*) fue creado para agrupar páginas web. Este algoritmo representa los documentos (páginas web) a partir de un vector de pesos $W_i = (w_{i1}, w_{i2}, \ldots, w_{in})$, donde cada valor w_{ik} es el peso asociado al término t_k. El peso de un término es 1 si el mismo pertenece al documento y 0 en otro caso. El número de términos utilizados para representar a los documentos se calcula utilizando un método de selección de variables aplicado a una muestra de la colección. Teniendo los documentos representados en este modelo este algoritmo construye iterativamente un árbol denominado DC-tree.

El DC-tree es un árbol que se contruye de forma iterativa insertando cada uno de los documentos. En este proceso de inserción, de forma similar al B^+-tree, se realizan operaciones de ajuste para mantener la estructura del árbol. Ver [80] para ver detalles de esta estructura.

Luego de construido el DC-tree, se realiza un proceso de búsqueda en amplitud sobre dicho árbol para identificar aquellos nodos que representen grupos "interesantes". Este criterio de interés se define teniendo en cuenta la cantidad de documentos almacenados en dicho nodo y el vector de pesos perteneciente al nodo. El agrupamiento resultante está formado por estos grupos interesantes. Los grupos que se obtienen de este algoritmo son disjuntos.

Tanto el STC como el DC-tree son algoritmos incrementales y pueden ser utilizados en colecciones en las cuales se adicionen documentos a través del tiempo; sin embargo, ambos presentan algunas deficiencias. En el caso del STC, la construcción del árbol de sufijos resulta extremadamente costosa respecto a la cantidad de términos de los documentos, por lo que sólo ha sido probado con documentos de pequeña dimensionalidad como los *snippets* que no sobrepasan las 35 palabras. Otros aspectos negativos en este algoritmo es que los grupos resultantes del mismo pueden presentar encadenamiento y que requiere prefijar valor para 1 parámetro.

En el caso del algoritmo DC-tree, el proceso de agrupamiento puede dejar documentos sin agrupar debido a los criterios definidos para considerar un grupo interesante. Otro aspecto negativo de este algoritmo es que requiere prefijar valores para 9 parámetros. Es importante mencionar además que el DC-tree puede resultar ineficaz para agrupar documentos con características diferentes a las páginas web y en los cuales el pesado binario puede ser inadecuado.

2.6. Algoritmos basados en densidad

Los algoritmos de este tipo, dado un conjunto de datos D, definen un criterio de densidad para un grupo y tratan de encontrar grupos o divisiones $C_1, C_2, \ldots, C_k$ de este conjunto de datos de forma tal que las densidades de estas divisiones sean las más cercanas posibles. En el mejor de los casos, estos algoritmos son capaces de detectar automáticamente el número de grupos k en los que se deben dividir los datos, así como son capaces

de descubrir grupos de diferentes formas y tamaños. Un ejemplo de estos algoritmos es el MajorClust [71].

En el algoritmo MajorClust el criterio de densidad de un grupo está asociado a la *atracción* que ejerce dicho grupo sobre los elementos que lo componen. Este criterio de atracción está relacionado con la cantidad de elementos del grupo así como también a la semejanza existente entre los elementos del mismo [42]. Formalmente, dado un grupo C y un documento d, la atracción que ejerce C sobre d se calcula como:

$$\sum_{p \in C} \varphi(p, d),$$

donde $\varphi(p, d)$ representa la semejanza existente entre el documento $p \in C$ y el documento d.

Este algoritmo, dada una colección de documentos D, contruye grupos disjuntos asignando cada documento d al grupo que ejerza sobre él la mayor atracción. MajorClust comienza formando un grupo unitario con cada documento, a partir de este punto comienza un proceso en el que iterativamente para cada documento d se calcula la atracción que ejerse cada grupo sobre él, teniendo en cuenta solamente aquellos elementos del grupo cuya semejanza con d sea mayor que cierto umbral t. Una vez que se calculan dichos valores se asigna d al grupo con mayor valor de atracción; si existen varios grupos en ese caso, se selecciona uno aleatoriamente. Este proceso se repite hasta que no hayan cambios en los grupos.

El pseudo-código del algoritmo MajorClust puede obsevarse en el Algoritmo 6. En este pseudo-código se asumió la existencia de una función $Cluster_{index}$ que, dado un documento d_i, devuelve el índice que ocupa (dentro del conjunto de grupos existentes) el grupo que contiene al documento d_i.

Este algoritmo tiene una complejidad computacional de $O(n^2)$ y aunque necesita del parámetro t, los autores comentan que puede asignársele siempre el valor 0,3. Sin embargo, este algoritmo es no determinista puesto que cuando existen varios grupos con el mismo valor de atracción se selecciona uno aleatoriamente. Esta deficiencia puede llevar a que ciertos grupos no sean correctamente identificados [71]. Por último, cabe decir que este algoritmo ha sido utilizado para agrupar colecciones de resúmenes [2] así como documentos de mayor tamaño [56].

2.7. Algoritmos basados en conjuntos frecuentes de ítems

La alta dimensionalidad de los documentos es una característica que hace muy costoso utilizar muchos de los algoritmos clásicos en el contexto del agrupamiento de documentos. Una alternativa a este problema son los agrupamientos basados en conjuntos frecuentes de ítems (FI, por sus siglas en inglés) [1]. Ejemplos de estos algoritmos son el *FTC* [6],[66] y el *FIHC* [17].

Algoritmo 6: MajorClust

Input: $D := \{d_1, d_2, \ldots, d_n\}$- document collection,
β - similarity threshold
Output: SC - Set of clusters

```
1  ready := false;
2  forall d_i ∈ D do SC := SC ∪ {d_i};
3  while ready == false do
4      ready := true;
5      forall d_i ∈ D do
6          index := Cluster_index(d_i);
7          index_1 := −1;
8          temp := 0;
9          forall C_j ∈ SC do
10             temp_1 := Σ_{d_k ∈ C_j} {Sim(d_k, d_i) | Sim(d_k, d_i) ≥ β ∧ d_k ≠ d_i};
11             if temp_1 > temp then
12                 temp := temp_1;
13                 index_1 := j;
14             end
15         end
16         "remove d_i from cluster C_index";
17         "insert d_i in cluster C_index_1";
18         ready := ready && (index == index_1)
19     end
20 end
```

El algoritmo FTC construye un conjunto de grupos disjuntos utilizando los conceptos de *cubrimiento de un FI*, *conjunto descriptivo de un agrupamiento* y la definición de una medida para estimar el solapamiento de un grupo.

Sea $F = \{F_1, F_2, \ldots, F_N\}$, el conjunto de todos los FI presentes en una colección de documentos D y que fueron calculados por algún algoritmo, como por ejemplo *Apriori* [1]. El cubrimiento de F_k se denota como $cov(F_k)$ y define al conjunto de documentos $d_j \in D$ que contienen a los ítems contenidos en F_k. Por otra parte, el conjunto descriptivo de un agrupamiento es el sub-conjunto $CD \subseteq F$, tal que se cumple la condición siguiente:

$$\bigcup_{F_k \in CD} cov(F_k) = D \tag{3}$$

Sea $f_j \in F$ un conjunto frecuente de ítems y C el grupo formado a partir de los documentos en $cov(f_j)$, una medida de solapamiento aplicada a C permite estimar la cantidad de conjuntos frecuentes que tiene en común éste con los grupos que se forman a partir del cubrimiento de cada conjunto $f_k \in F$, $k = 1...N, k \neq j$. Mientras menos conjuntos frecuentes contienen los documentos $d \in C$ menor será el valor de esta medida y por consiguiente, el solapamiento de este grupo. Las medidas que se han definido para calcular dicho solapamiento son el *Standard overlap* (SO) y el *Entropy overlap* (EO) [6]

así como también el *Improve Standard overlap* (ISO) y el *Improve Entropy overlap* (IEO) [66]. Las fórmulas que definen a las funciones anteriores pueden observarse en la tabla 2.

Tabla 2. Funciones para el cálculo del solapamiento entre grupos

Función	Fórmula
SO	$\frac{\sum_{d_j \in C_i}(f_j - 1)}{\lvert C_i \rvert}$
EO	$\sum_{d_j \in C_i} -\frac{1}{f_j} \ln \frac{1}{f_j}$
ISO	$\sum_{d_j \in C_i}(f_j - k)$, donde $k = 2^n - 1$
IEO	$\sum_{d_j \in C_i} -\frac{1}{m \cdot f_j} \ln \frac{1}{m \cdot f_j}$, donde $m = \sum_{d_j \in C_i}(\frac{1}{f_j})$

Utilizando lo anteriormente mencionado, FTC construye iterativamente un conjunto de grupos $SC = \{C_1, C_2, \ldots, C_M\}$ de forma tal que cada grupo adicionado tenga el menor solapamiento respecto a los grupos que se pueden formar con los FI que no se han utilizado en iteraciones anteriores. Para construir los grupos se parte del conjunto $F = \{F_1, F_2, \ldots, F_N\}$ e iterativamente se selecciona el F_k cuyo grupo formado a partir de los documentos $d_j \in cov(F_k)$ tenga el menor valor de solapamiento. Luego de esta selección, se elimina de F el elemento F_k y se adiciona a SC el grupo formado por los documentos $d_j \in cov(F_k)$ que no pertenezcan a algún grupo ya calculado.

El pseudo-código del algoritmo FTC puede obsevarse en el Algoritmo 7. En este pseudo-código se asumió la existencia de las funciones:

a) Apriori, que calcula, dado un conjunto de transacciones[4], el conjunto de FI presentes en la misma.
b) *Cov*, que calcula el cubrimiento de un FI dado.
c) $Overlap_m$, que calcula el cubrimiento del grupo formado por un FI dado.

El algoritmo FIHC permite obtener un conjunto de grupos disjuntos y a partir de este conjunto puede obtenerse además una jerarquía de grupos. Este algoritmo se apoya en los conceptos de *ítem frecuente global* e *ítem grupo-frecuente*. El concepto de ítem frecuente global es el mismo que se utiliza para definir a un ítem frecuente en [1]. Un ítem f es grupo-frecuente respecto al grupo C_i si, dado un umbral α, el ítem está contenido en al menos α documentos que pertenecen a c_i.

Para obtener el conjunto de grupos disjuntos, este algoritmo parte del conjunto $F = \{F_1, F_2, \ldots, F_M\}$ de todos los FI de la colección. Como primer paso se contruye un conjunto de grupos $SC = \{C_1, C_2, \ldots, C_M\}$ donde cada $C_i = cov(F_i)$, $i = 1...M$. Como se puede notar los grupos formados pueden ser solapados. Debido a lo anterior, en el segundo paso se procede a asignar cada documento $d_j \in D$ en el grupo más adecuado.

[4] En este caso, cada transacción es la descripción de un documento

Algoritmo 7: FTC

Input: $D := \{d_1, d_2, \ldots, d_n\}$ - document collection,
α - minimum support
Output: SC - Set of clusters,
SF - Set of FI selected

$SF := \emptyset$;
$SC := \emptyset$;
$temp := D$;
$n := |D|$;
$RF := \texttt{Apriori}(D, \alpha)$;
`// forming the set of clusters`
while $\left|\bigcup_{F \in SF} cov(F)\right| \neq n$ **do**
  $FBest := \arg min\{Overlap_m(F), F \in RF\}$;
  $SF := SF \cup FBest$;
  $RF := RF \setminus FBest$;
  $SC := SC \cup (cov(FBest) \cap temp)$;`// forming the cluster`
  $temp := temp \setminus cov(FBest)$;
end

Un grupo C_i es adecuado para contener un documento d_j si existen muchos ítems globalmente frecuentes en d_j que son grupo-frecuentes en C_i. La siguiente fórmula es utilizada para calcular el índice de pertenencia de un documento a un grupo:

$$Score(C_i, d_j) = \left(\sum_{x} n(x) \cdot s - grupo(x)\right) - \left(\sum_{x'} n(x') \cdot s - global(x')\right), \qquad (4)$$

donde x representa a aquellos ítems en d_j que son frecuentes globalmente y que a la vez son grupo-frecuentes respecto a C_i, x' representa a aquellos ítems en d_j que son frecuentes globalmente y que no son grupo-frecuentes respecto a C_i, $n(x)$ y $n(x')$ representan la frecuencia de los ítems x y x' en d_j.

Utilizando (4) se determina para cada documento d_j el grupo con mayor índice de pertenencia, se asigna d_j a dicho grupo y se elimina del resto de los grupos. En el caso de que existan varios grupos con el mismo valor de índice, se selecciona aquel que fue formado por el FI de mayor cardinalidad. El grupo que resulte vacío producto de la operación anterior es eliminado.

Luego de obtener este conjunto de grupos disjuntos, se puede construir una jerarquía de grupos utilizando una estrategia aglomerativa que parte de los grupos formados por los FI de mayor cardinalidad o longitud y que utiliza una variante de la función (4) [17].

El pseudo-código del algoritmo FIHC se puede observar en el Algoritmo 8. En este pseudo-código se asumió además de la función Apriori, comentada en el algoritmo anterior, la existencia de las siguientes funciones:

a) Cluster-frequent, que determina si un FI es grupo-frecuente en un grupo determinado.

b) BuildTree, que construye una jerarquía utilizando el conjunto de grupos detectados.
c) Pruning_Merging, que realiza un proceso de poda sobre la jerarquía construida.

Algoritmo 8: FIHC

Input: $D := \{d_1, d_2, \ldots, d_n\}$ - document collection,
α - minimum global support,
β - minimum cluster support
Output: SC - Set of clusters

$SC := \emptyset$;
$F :=$ `Apriori(`D`,`α`)`;
`// Building initial clusters and the set of cluster-frequent items for each cluster`
forall $f_i \in F$ **do** $SC := SC_i \cup cov(f_i)$;
forall $C_i \in SC$ **do** $C_i.CFI := \{f \mid f \in F \wedge$ `Cluster-frequent(`C_i`,`β`)`$\}$;
`// Making clusters disjoint`
forall $d_j \in D$ **do**
    $k := \arg max_i\{$`Score(`C_i`,` d_j`)` $\mid C_i \in SC\}$;
    "Delete d_j from all clusters";
    "Add d_j to cluster C_i";
end
"Update the set of cluster-frequent items for each cluster in SC";
`// Building the Cluster Tree`
$T :=$ `BuildTree(`SC`)`;
`Pruning_Merging(`T`);` `// Tree pruning`

La principal deficiencia de estos métodos es que los grupos que se obtienen dependen tanto del valor de umbral de soporte mínimo para el cálculo de los FI, así como (en el caso del FIHC) del umbral definido para determinar los ítems grupo-frecuentes. Además, los grupos que se obtengan para un cierto valor de los umbrales dependen del orden de análisis de los FI en el caso del FTC, así como del orden de los documentos en el caso del FIHC.

Otro aspecto negativo de estos algoritmos, que no es muy frecuente, es que pueden dejar documentos sin agrupar. Nótese que pueden existir documentos que no contengan a ningún término frecuente y por tanto, ambos algoritmos no los incluirán como parte de la solución.

2.8. Algoritmos basados en técnicas genéticas

Los algoritmos genéticos son heurísticas que ayudan a solucionar problemas que por vías analíticas serían difíciles de resolver debido al costo computacional involucrado. Este tipo de heurísticas tratan de emular el comportamiento evolutivo de los seres vivos a través de procesos como cruzamiento, mutación o selección natural [30]. Ejemplos de algoritmos de agrupamiento de documentos que utilizan estas técnicas o heurísticas se pueden apreciar en [12] y [69].

El algoritmo desarrollado en [12] determina dinámicamente el número de grupos para particionar la colección de documentos. Para realizar este paso se selecciona un subconjunto o muestra de los documentos y si la cantidad de documentos de ésta es menor que 15, se aplica el algoritmo de Calinski y Harabasz [10], en otro caso se aplica un algoritmo genético. El algoritmo genético se aplica para determinar el número de grupos óptimo y el mismo parte del árbol de cubrimiento mínimo que se puede obtener a través del grafo de semejanza que se forma con los documentos de la muestra.

Una vez que se determina el número de grupos a obtener, éstos se construyen utilizando algún método de agrupamiento de la librería de métodos CLUTO (Single-link, Complete-link o UPGMA)[35]. El pseudo-código de este algoritmo puede observarse en el Algoritmo 9. En este pseudo-código se asumió la existencia de las funciones:

a) Cal-Har, que invoca al algoritmo de Calinski y Harabasz para el cálculo del número de grupos a formar.
b) *GenAlgo*, que invoca al algoritmo genético utilizado en el cálculo del número de grupos a formar si el tamaño de la muestra es mayor que 15.
c) *CLUTO*, que invoca a uno de los algoritmos de agrupamientos (Single-link, Complete-link ó UPGMA) contenidos en la esta librería.

Algoritmo 9: Genético

Input: $D := \{d_1, d_2, \ldots, d_n\}$ - document collection
Output: SC - Set of clusters,
SF - Set of FI selected

```
1 "Build an aleatory sample set R from set of documents D";
  // Determining the number of clusters to be formed
2 if |R| < 15 then k := Cal-Har(R);
3 else k := GenAlgo(R);
  // Building the k clusters
4 SC := CLUTO(D,k);
```

En [69] se desarrolla un algoritmo llamado MVGA, el cual calcula dinámicamente el número óptimo de grupos en los que se debe agrupar la colección y durante este proceso obtiene también dicho conjunto de grupos. Para obtener dicho conjunto de grupos, este algoritmo se basa en una estrategia similar a la del algoritmo k-means y en una implementación de un algoritmo genético. En este algoritmo, cada individuo se representa como un vector de elementos, donde cada elemento representa el centroide de un grupo del agrupamiento a obtener. Estos centroides son seleccionados aleatoriamente del conjunto de documentos de la colección, los cuales son previamente representados utilizando el modelo de Espacio Vectorial.

Partiendo de una población de 100 individuos y utilizando como función de aptitud el índice DB [16], este algoritmo realiza un proceso evolutivo mediante los operadores de

selección, cruzamiento y mutación con los cuales se exploran al mismo tiempo diferentes formas de particionar al conjunto de documentos. Al ejecutar este proceso un cierto número de iteraciones o al alcanzar la condición de convergencia, el individuo con mejor valor de la función aptitud y que pertenece a la última generación, representa el agrupamiento óptimo entre todos los explorados. Los pasos que forman parte del funcionamiento del algoritmo MVGA están descritos en [69].

Estos algoritmos presentan varias deficiencias, una de ellas es que resultan costosos cuando trabajan con datos de alta dimensionalidad, ya que el proceso evolutivo se vuelve muy complejo. Debido a esto, en el caso del algoritmo MVGD se realizó un proceso de selección de términos para reducir la dimensionalidad, es importante notar que este proceso puede afectar la calidad del agrupamiento resultante. Otro aspecto negativo de este algoritmo es que necesita para su ejecución asignar valores a 5 parámetros.

Una deficiencia notable en el algoritmo desarrollado en [12] es que el proceso de determinar dinámicamente el número de grupos depende en gran medida de la calidad de la muestra seleccionada, si ésta no es representativa de la colección, entonces el número de grupos calculados puede no ser el más idóneo. Otra deficiencia de este algoritmo es que sólo utiliza 10 individuos para conformar la población inicial, por lo que la diversidad de ésta puede resultar baja y por consiguiente, aumenta la posibilidad de que el algoritmo converja a un óptimo local o que aumente el tiempo de convergencia al óptimo global.

Por último, es importante comentar que los experimentos realizados con cada uno de los algoritmos anteriores utilizaron colecciones pequeñas de documentos que contenían 200 documentos como máximo, por lo que es muy posible que resulten ineficientes o incluso imposibles de aplicar con respositorios de documentos de mayor tamaño.

2.9. Algoritmos basados en factorización de matrices

Teniendo en cuenta el modelo de Espacio Vectorial [64], una colección de documentos puede representarse como una matriz $X_{m\times n}$, donde n representa la cantidad de documentos de la colección y m la cantidad de términos presentes en ésta. Cada elemento X_{ji} representa el peso del término t_j en el documento d_i, o sea, un valor mayor o igual a cero, por lo que se puede decir que X es una matriz no negativa.

Los algoritmos basados en factorización de matrices, y en específico de matrices no negativas (como el caso de X), se basan en conceptos del álgebra vectorial. Asumiendo que en la colección hay presentes k tópicos o grupos, cada uno expresado a través de un vector de términos, un documento d_i de la colección pudiera verse como una combinación lineal de estos vectores de tópicos si consideramos el espacio semántico de dimensión k del cual este conjunto de vectores es base.

Dado un valor k que representa el número de grupos disjuntos en los que se desea agrupar una colección V, un algoritmo de este tipo aproxima la matriz no-negativa X que representa a V mediante el producto de dos matrices no-negativas $W_{m\times k}$ y $H_{k\times n}$. Cada

columna de W contiene al vector que define a un grupo presente en la colección mientras que cada columna de H contiene el vector de coeficientes utilizado para aproximar el vector de la correspondiente columna de X. Una vez que se tienen estas matrices, se recorre cada documento d_j asignándolo al grupo que más aportó en la combinación lineal que forma a dicho documento.

Un ejemplo de algoritmos de este tipo se puede apreciar en los trabajos de Xu *et al.* [82] y los de Shahnaz [65]. Ambos métodos parten de inicializar las matrices W y H con valores no negativos y a partir de este punto realizan un proceso en el cual iterativamente se va ajustando los valores de W_{ji} y H_{ji} mediante transformaciones sucesivas. Los métodos son diferentes en cuanto a formas de actualizar los valores de cada matriz. La complejidad computacional de estos algoritmos es $O(tkn)$ y $O(tkmn)$, donde t es el número de iteraciones realizadas.

El pseudo-código del algoritmo NMF puede observarse en el Algoritmo 10. En este pseudo-código se asumió que existen funciones para determinar el número de filas (Rows) y columnas (Columns) de una matriz.

Algoritmo 10: NMF

Input: $D := \{d_1, d_2, \ldots, d_n\}$ - document collection,
k - number of groups to form
Output: SC - Set of clusters

```
1  "Build terms matrix X from collection D";
2  "Initialize matrices W and H with random non-negative numbers";
3  while stop-conditions (a) or (b) are not satisfied do
       // Updating matrices W and H
4      for i = 1 to Rows(X) do
5          for j = 1 to k do
6              for l = 1 to Columns(X) do
                   // Updating W_ij y H_jl
7                  W_ij = W_ij (XH^t)_ij / (UHH^t)_ij;
8                  H_jl = H_jl (X^t H)_jl / (HW^t W)_jl;
9              end
10         end
11     end
12 end
13 "Normalize both W y H";
14 "Build set of clusters using matrix H;
```

La principal deficiencia de estos algoritmos es que el resultado del agrupamiento depende de qué tan buena sea la aproximación realizada de V. Si durante el proceso de aproximación no se llegó a la condición de convergencia sino que dicho proceso se detuvo al realizar un número de iteraciones prefijado, es muy posible que la calidad del agru-

pamiento sea baja. Otro aspecto a señalar es que este algoritmo necesita fijar *a priori* el número de grupos a obtener por lo que el resultado depende de este valor.

2.10. Algoritmos basados en grafos

Los algoritmos de agrupamiento de documentos basados en grafos representan la colección de documentos a través de un grafo $G = \langle V, E, w \rangle$, en el cual los vértices representan a los documentos y las aristas están etiquetadas con el valor de semejanza existente entre los vértices que la componen. Una vez representada la colección, estos algoritmos construyen un cubrimiento de este grafo utilizando propiedades de la Teoría de Grafos. Los grupos resultantes de este algoritmo estarán formados por los elementos de este cubrimiento.

Algunos de los algoritmos de agrupamiento de documentos que utilizan esta técnica son: GLC [68], Compacto Incremental [58], Fuertemente Compacto Incremental [60], Star [3],[4], Extended Star [21],[22] y Generalized Star [63]. Todos estos algoritmos varían respecto al tipo de grafo que utilizan, a la heurística utilizada para obtener el cubrimiento de dicho grafo e incluso respecto al tipo de cubrimiento que obtienen. Con el objetivo de describir estos algoritmos se hace imprescindible introducir algunos conceptos básicos.

Conceptos básicos

Sea $D = \{d_1, d_2, \ldots, d_N\}$ una colección de documentos y S una función de semejanza simétrica entre documentos; es decir, para todo par de documentos d_i y d_j se cumple que $S(d_i, d_j) = S(d_j, d_i)$. Un *grafo de semejanza* $G = \langle V, E, w \rangle$, es un grafo no dirigido etiquetado en el cual los vértices representan a los documentos de la colección, cada arista (d_i, d_j) representa la semejanza existente entre los documentos que la componen y w es la función que asigna los pesos a las aristas.

Sea $\beta \in \Re$, tal que $\beta \in [0, 1]$, un umbral de semejanza y G un grafo de semejanza. Un *grafo de β-semejanza* se denota por $G_\beta = \langle V, E_\beta \rangle$ y es el grafo no dirigido que se obtiene a partir de G si se eliminan todas las aristas $e \in E$ tal que $w(e) < \beta$.

Dado el grafo $G_\beta = \langle V, E_\beta \rangle$, dos documentos d_i, d_j representados por los vértices $v_i, v_j \in V$ serán denominados *β-semejantes* si existe la arista $(v_i, v_j) \in E_\beta$. Los vértices v_i para los que no exista $v_j \in V, v_j \neq v_i$ tal que v_i y v_j sean β-semejantes, serán denominados *β-aislados*.

Un *grafo de máxima semejanza* se denota por $G_{max} = \langle V, E \rangle$ y es el grafo dirigido donde los vértices representan a los documentos $d_i \in D$ y donde existe una arista $(v_i, v_j) \in E$, si se cumple que $Sim(d_i, d_j) = max_{d_p \in D} Sim(d_i, d_p)$.

Dado un grafo dirigido $G = \langle V, E \rangle$, se llamará *Clausura-O de G* al grafo no dirigido $\overline{G} = \langle V, \overline{E} \rangle$, tal que para cada par de vértices $v_i, v_j \in V$ existe una arista no dirigida $(v_i, v_j) \in \overline{E}$, si y sólo si $(v_i, v_j) \in E$ ó $(v_j, v_i) \in E$.

Algoritmo GLC

Uno de los casos más simples de algoritmos basados en grafos es el algoritmo GLC, el cual obtiene de forma incremental las componentes conexas del grafo de β-semejanza, las cuales representan a los grupos que se desean obtener.

El algoritmo GLC realiza un proceso en el cual cada documento d_i se analiza una sola vez, determinándose para éste cuáles son los documentos pertenecientes a los grupos existentes que son β-semejantes con él. Si el documento d_i no tiene elementos β-semejantes entonces se crea un nuevo grupo unitario que contiene a d_i. En caso contrario todos aquellos grupos que contienen algún documento β-semejante con d_i se unen y forman un nuevo grupo en el que también se adiciona d_i.

El pseudo-código del algoritmo GLC puede observarse en el Algoritmo 11. En este pseudo-código se asumió la existencia de una función *Connected* que dado un documento, un grupo y un umbral, determina si dicho documento está conectado con el grupo.

Algoritmo 11: GLC

Input: $D := \{d_1, d_2, \ldots, d_n\}$ - document collection,
β - similarity threshold
Output: SC - Set of clusters

```
1  SC := ∅;
2  L := ∅; // Lis t of clusters to be joined
3  forall d_i ∈ D do
       // Determining the clusters connected with d_i
4      forall C_j ∈ SC do
5          if Connected(d_i, C_j, β) then L := L ∪ C_j;
6      end
7      if L = ∅ then SC := SC ∪ {d_i};
8      else
9          "remove from SC all clusters belonging to L";
10         C := {d_i};
11         forall C_j ∈ L do "Add documents in C_j to cluster C";
12         SC := SC ∪ C;
13     end
14 end
```

La complejidad computacional de este algoritmo es $O(n^2)$ ya que para cada documento es necesario calcular la semejanza de éste con los restantes. Sin embargo, es importante mencionar que en la práctica este algoritmo en muchos casos no requiere calcular siempre todas estas semejanzas. Lo anterior se debe a que cuando se encuentra un documento d_j que es β-semejante con d_i, ya no es necesario calcular la semejanza de d_i con el resto de los documentos que pertenecen al mismo grupo que d_j.

El resultado del algoritmo GLC es independiente del orden de análisis de los documentos y no impone restricciones a la forma de los grupos obtenidos. Su mayor limitación es

que las componentes conexas sobre el grafo de β-semejanza presentan un elevado efecto de encadenamiento por lo que pueden obtenerse grupos poco cohesionados; es decir, grupos que incluyan documentos muy poco semejantes.

Algoritmo Compacto Incremental

Este algoritmo, dado el grafo de máxima semejanza $G_{max} = \langle V, E \rangle$ que representa la colección de documentos D, obtiene de forma incremental un conjunto de grupos disjuntos $G = \{G_1, G_2, \ldots, G_K\}$ en el que cada G_i es un *conjunto compacto*. Los conjuntos compactos coinciden con las componentes conexas del grafo $\overline{G}_{max}$ [57].

Cada vez que se inserta un objeto O en el grafo, se calcula la semejanza de éste con los objetos existentes y se actualiza G_{max}. Esta actualización involucra eliminar aristas existentes en G_{max} y adicionar otras nuevas, por lo tanto puede producir cambios en el agrupamiento, ya que pueden surgir nuevos conjuntos compactos y otros conjuntos compactos existentes pueden dejar de serlo. Debido a lo anterior, luego de que se produce la actualización del grafo, se realiza un proceso de reconstrucción de los conjuntos compactos a partir del objeto O y de aquellos grupos que pueden dejar de ser compactos.

Los grupos a tener en cuenta en este proceso son aquellos que están conectados con el objeto O. Un grupo G está conectado con O si existe un objeto $O' \in G$ que cumple que O es el objeto más β-semejante de O' o viceversa. De lo anterior podemos deducir que aquellos grupos que no estén conectados con O no pierden la propiedad de ser compactos y que si no existe ningún grupo conectado con O, entonces el grupo formado únicamente por el propio O es un conjunto compacto.

Una vez identificados los grupos conectados con O, el algoritmo construye los siguientes conjuntos:

- ***Grupos a Procesar***. Estos grupos representan a aquellos conjuntos compactos que pueden perder la propiedad de ser compactos y que, por consecuencia, deben ser reconstruidos. Estos grupos son eliminados de la lista de grupos existentes.
- ***Objetos a Unir***. A este conjunto pertenecen todos aquellos objetos O' que tienen como único objeto más β-semejante a O y que antes de la inserción de éste eran β-aislados o tenían solamente un objeto más β-semejante. Estos objetos son adicionalmente eliminados de los grupos a los que pertenecían.
- ***Grupos a Unir***. Un grupo G_i pertenece a este conjunto si no pertenece a *Grupos a Procesar* y cumple además que: (i) está conectado con O y (ii) la componente conexa que lo representa en $\overline{G}_{max}$ no perdió ninguna arista producto de la inserción de O. Los elementos de estos conjuntos pertenecen a la misma componente conexa de O en $\overline{G}_{max}$ y por lo tanto a su mismo conjunto compacto. Estos grupos son eliminados de la lista de grupos existentes.

Una vez construidos estos conjuntos, se forma un nuevo grupo compacto que contiene al objeto O, a todos los objetos en *Objetos a Unir* y a los objetos contenidos en la unión de

los grupos de *Grupos a Unir*. Posteriormente se procesan cada uno de los grupos insertados en *Grupos a Procesar* para reconstruir los conjuntos compactos.

Para cada grupo $g_i \in$ *Grupos a Procesar*, se sigue un proceso iterativo en el cual se selecciona un objeto $O' \in g_i$ y se construye su componente conexa en $\overline{G}_{max}$; esta componente conexa es un nuevo conjunto compacto y por lo tanto se adiciona al conjunto de grupos determinados. Este proceso termina cuando todos los objetos de g_i pertenecen a algún conjunto compacto. El conjunto de pasos que realiza este algoritmo están descritos en [57].

La complejidad computacional de este algoritmo es de $O(n \cdot (e + nm)))$, donde $e = |E|$, $n = |V|$ y m es la cantidad de rasgos que describen a los objetos. No obstante a lo anterior y considerando que en la práctica la cantidad de objetos más semejantes a un objeto dado es pequeña (1 en muchos casos) se toma $e = cn$, donde c es la cantidad máxima de objetos más semejantes a uno dado. Teniendo en cuenta la consideración anterior la complejidad computacional de este algoritmo es $O(n^2)$.

Por último, es válido mencionar que los agrupamientos construidos por el algoritmo compacto incremental son independientes del orden de análisis los documentos. Sin embargo, este algoritmo presenta algunas deficiencias: (i) por lo general, la cantidad de grupos que se obtienen es grande, (ii) los grupos pueden tener encadenamiento y ser poco cohesionados, aunque en menor magnitud que los grupos generados por el GLC.

Algoritmo Fuertemente Compacto Incremental

Este algoritmo también representa la colección de documentos a través de su grafo de máxima semejanza pero, a diferencia del algoritmo Compacto Incremental, permite obtener un conjunto de grupos que pueden ser solapados y que son relativamente pequeños y densos [60],[57].

El algoritmo Fuertemente Compacto Incremental basa su funcionamiento en la relación existente entre los conjuntos compactos y los fuertemente compactos, la cual establece que *"Todo conjunto compacto es la unión finita de conjuntos fuertemente compactos"* [46]. Utilizando la relación anterior, los conceptos de *grafo reducido según la relación de fuerte-conexidad* y *base de un grafo dirigido*, este algoritmo construye un conjunto de grupos solapados en los que cada grupo es un conjunto fuertemente compacto en G_{max}.

Se llamará *base del grafo dirigido* $G = \langle V, E\rangle$ al conjunto de vértices $B \subseteq V$, tal que el mismo satisface las condiciones siguientes [34]:

1. Para todo $v_i \in V$ existe un camino desde un vértice en B a v_i.
2. El conjunto B es el menor que cumple esta propiedad.

Sea $G = \langle V, E\rangle$ un grafo dirigido y $CC = \{CP_1, CP_2, \ldots, CP_k\}$ el conjunto de componentes fuertemente conexas que lo forman; se llamará *grafo reducido según la fuerte-conexidad de* G al grafo denotado por $G_r/f\text{-}C$, cuyos vértices son las componentes fuerte-

mente conexas de G[5] y en el cual existe una arista del vértice CP_l a otro CP_m si existe al menos una arista de un vértice $v \in CP_l$ a otro vértice $v' \in CP_m$.

De forma similar al algoritmo Compacto Incremental, cada vez que se inserta un nuevo elemento O en $\overline{G}_{max}$ se calcula la semejanza de éste con todos los objetos existentes y como consecuencia se actualiza el grafo. Posteriormente se reconstruyen los conjuntos compactos y, a partir de éstos, los conjuntos fuertemente compactos. Para lograr esto, para cada sub--grafo determinado por los conjuntos compactos, se construye el grafo reducido asociado y su correspondiente base. Una vez que se tiene la base del grafo reducido, los conjuntos fuertemente compactos se construyen utilizando los vértices que componen dicha base. Los pasos de este algoritmo se encuentran descritos en [57].

Este algoritmo es incremental, su complejidad computacional es $O(n^2)$, no depende del orden de análisis de los documentos, no necesita conocer el número de grupos a determinar y estos grupos obtenidos pueden tener formas arbitrarias. La principal limitante de este algoritmo es que obtiene muchos grupos y generalmente éstos contienen pocos elementos.

Algoritmo Star

El algoritmo Star fue propuesto por Aslam [3],[4] y ha sido utilizado en varias aplicaciones relacionadas con el filtrado y organización de información. Este algoritmo construye a partir del cubrimiento del grafo G_β utilizando *sub-grafos en forma de estrella*, un conjunto de grupos SC que pueden ser solapados; en este cubrimiento, cada sub-grafo determina un grupo de G. El cubrimiento realizado por el algoritmo Star es similar al cubrimiento que se obtiene si se utilizaran cliques (el cubrimiento a partir de cliques es un problema NP-completo [86]); de esta forma, Star construye un conjunto de grupos menos cohesionados que los que se formarían utilizando los cliques, pero con una complejidad computacional mucho menor.

Un *sub-grafo en forma de estrella*, es un sub-grafo de $m+1$ vértices, en el cual existe un vértice llamado "centro", m vértices denominados "satélites" y se cumple que: (i) el centro tiene un grado mayor o igual que el resto de los vértices del sub-grafo y (ii) existe una arista del centro a cada uno de los satélites. El problema de encontrar los sub-grafos en forma de estrella se reduce al problema de determinar el conjunto X de vértices centro.

El pseudo-código del algoritmo Star se puede observar en el Algoritmo 12.

El algoritmo Star sigue una estrategia *greedy*. Como primer paso, se construye el grafo $G_\beta = \langle V, E_\beta \rangle$ y crea una lista L con todos los vértices del grafo ordenados de mayor a menor respecto a su grado. Este ordenamiento permite seleccionar primero los vértices centro que forman sub-grafos más densos. Seguidamente se realiza un proceso que selecciona iterativamente el vértice $v \in L$ de mayor grado que no pertenezca a ningún sub-grafo determinado en iteraciones anteriores. Luego, el vértice v se adiciona al conjunto X y se

[5] Los elementos del conjunto $CC = \{CP_1, CP_2, \ldots, CP_k\}$.

Algoritmo 12: Star

Input: $D := \{d_1, d_2, \ldots, d_n\}$ - document collection,
β - similarity threshold
Output: SC - Set of clusters

```
1  "Build G_β = ⟨V, E_β⟩ from D";
2  L := V;
3  X := ∅;
4  while L ≠ ∅ do
5      d := arg máx{|d_i.Adj| , d_i ∈ L};
6      X := X ∪ {d};
7      "Remove d and its adjacent vertices from L";
8  end
9  SC := ∅;
10 forall center c ∈ X do
11     SC := SC ∪ {{c} ∪ c.Adj};
```

elimina de L junto con sus vértices adyacentes. Este proceso se repite hasta que lista L queda vacía, garantizando que cada vértice pertenezca al menos a un sub-grafo.

Este algoritmo no necesita conocer el número de grupos a obtener sino que lo descubre a través de las propias características de los documentos de la colección. Es importante señalar que, aún cuando el algoritmo Star depende de la selección del valor de β, este parámetro sólo representa el valor mínimo de semejanza que deben tener dos documentos para aceptar que éstos son semejantes.

La complejidad computacional del algoritmo Star es $O(n^2)$ y el mismo garantiza una semejanza de al menos β entre los centros y sus respetivos satélites de cada sub-grafo (grupo) determinado. Aunque entre dos satélites de un mismo sub-grafo no se garantiza este valor de semejanza, en [4] se exponen resultados que argumentan que el valor de esta semejanza es relativamente alta y que los sub-grafos de este tipo conforman grupos con una semejanza promedio relativamente elevada. No obstante a lo anterior, este algoritmo tiene algunas deficiencias.

La primera deficiencia radica en que el agrupamiento resultante depende del orden de análisis de los vértices. Esta deficiencia se pone de manifiesto en el momento en que se tiene que seleccionar el vértice de mayor grado para ser incluido en el conjunto X; si en este momento existe más de un vértice con el mismo grado y éstos son adyacentes, el agrupamiento resultante puede variar. Esta situación se expone en los grafos (A) y (B) de la figura 1, en la cual los vértices negros representan a aquellos incluidos en X.

Como se puede observar en la figura 1, el grafo (A) muestra el agrupamiento resultante si el algoritmo analiza primero el vértice 5. Sin embargo, si se analizan primero cualquiera de los vértices 2 ó 7, el conjunto de grupos que se obtiene difiere notablemente como se puede observar en el grafo (B). Es importante mencionar que este ejemplo no sólo muestra cuan diferentes pueden ser los agrupamientos de un algoritmo que sea dependiente

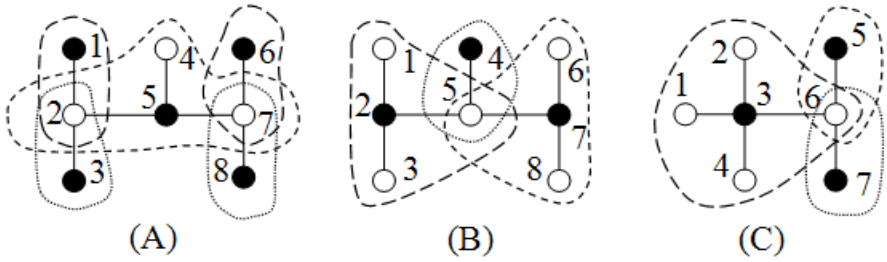

Figura 1. Deficiencias del algoritmo Star

del orden, sino cómo esta dependencia puede afectar la calidad del agrupamiento al no permitir que se descubran grupos densos.

La segunda deficiencia de este algoritmo es que, independientemente del orden en que se procesen los elementos, se pueden obtener grupos "ilógicos". En este reporte técnico se entenderá que un grupo g_1 es ilógico si cumple las siguientes condiciones:

- Existe un elemento $e \in g_i$ que es más denso[6] que el vértice centro c que define a g_1.
- El elemento e puede agrupar, si se considera como centro, a los vértices que son agrupados sólo por el centro c.

Esta limitación se debe a que el algoritmo no permite que dos vértices centro sean adyacentes. Como se puede observar en el grafo (C) de la figura 1, el vértice 6 debería haber sido seleccionado como centro (ser incluido en X) y no sus correspondientes adyacentes menos densos.

Algoritmo Extended Star

El algoritmo Extended Star fue desarrollado por Gil *et al.* [21] con el objetivo de solucionar las deficiencias detectadas en Star. Este algoritmo sigue una heurística semejante a la que aplica Star para la construcción del agrupamiento de G_β, pero a diferencia de este último, define un nuevo concepto de *centro* que determina un conjunto de vértices X diferente al que obtiene Star y por consecuencia, un conjunto de grupos (sub-grafos) diferentes.

Un vértice $v \in V$ se considera como *centro* si tiene un adyacente v', tal que en éste se satisface alguna de las siguientes condiciones:

- v' no tiene vértices centro adyacentes.
- Si se considera a v'' como el vértice centro de mayor grado que es adyacente a v', entonces se cumple que v'' tiene un grado no mayor al de v.

La heurística que sigue este algoritmo, para determinar el conjunto de vértices X que definen el agrupamiento, utiliza además del grado de los vértices el *grado complemento* de

[6] El criterio de densidad que utiliza el algoritmo Star tiene en cuenta solamente el grado del vértice centro del sub-grafo que forma al grupo

éstos. Dado un vértice v, su *grado complemento* se denota como $v.GC$ y es la cantidad de adyacentes que tiene v que no pertenecen a ninguno de los grupos formados a partir de los vértices insertados en el conjunto X hasta el momento.

El pseudo-código del algoritmo Extended Star se puede observar en el Algoritmo 13.

Algoritmo 13: Extended Star

Input: $D := \{d_1, d_2, \ldots, d_n\}$ - document collection,
β - similarity threshold
Output: SC - Set of clusters

```
1  "Build G_β = ⟨V, E_β⟩ from D";
2  SC := {v | |v.Adj| = 0};
3  L := V \ SC;
4  "Calculate the complement degree of each vertex in G_β";
5  while a non clustered vertex v ∈ V exist do
6      M_0 := {v | v ∈ L ∧ v.CD = arg máx{d_i.CD, d_i ∈ L}};
7      M := {v | v ∈ M_0 ∧ |v.Adj| = arg máx{|d_i.Adj|, d_i ∈ M_0}};
8      forall vertex c ∈ M do
9          if c satisfies the condition to be center then
10             C := {c} ∪ c.Adj;
11             if C ∉ SC then SC := SC ∪ {C};
12         end
13     end
14     L := L \ M;
15     "Update the complement degree of each vertex in L";
16 end
```

El funcionamiento del algoritmo Extended Star tiene como primer paso la construcción de G_β, para lo cual se calcula la semejanza entre todo par de vértices. Posteriormente se insertan en X todos los vértices aislados y se construye una lista L con el resto de los vértices del grafo. Una vez formada L se comienza un proceso en el cual, iterativamente se selecciona el sub-conjunto $M \subseteq L$ de vértices con mayor grado complemento y de éste último, el sub-conjunto $M' \subseteq M$ de mayor grado. Cada uno de los elementos incluidos en M' son procesados y aquellos que cumplan las condiciones para ser centro y formen un grupo diferente a los determinados hasta el momento, son adicionados a X. A continuación se eliminan los elementos de M' de la lista L, se actualiza el grado complemento de los vértices que quedaron en L y se repite el proceso hasta que todos los elementos estén agrupados. Este criterio de parada puede hacer que el algoritmo caiga en un ciclo infinito, dejando de agrupar elementos; incluso si se modifica el criterio de parada para evitar el ciclo infinito, el algoritmo puede dejar elementos sin agrupar. Esta situación se ilustra en el grafo (A) de la figura 2.

La complejidad computacional de este algoritmo es $O(n^2)$ y el conjunto de grupos que se obtiene es independiente del orden de análisis de los vértices en G_β. Sin embargo, el

algoritmo Extended Star tiene algunas deficiencias considerables. La primera deficiencia es que puede dejar elementos sin agrupar.

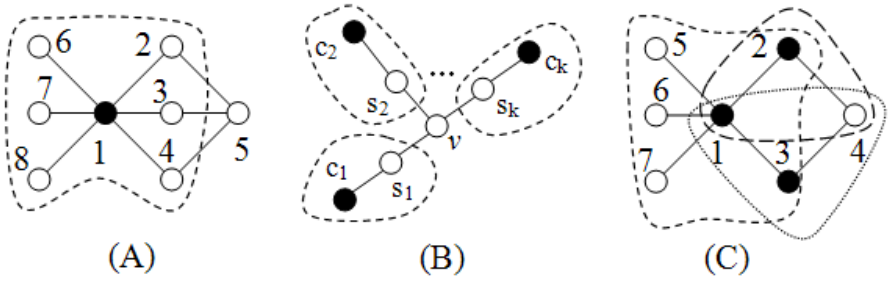

Figura 2. Deficiencias del algoritmo Extended Star

Para el grafo (A) de la figura 2, el algoritmo realiza el siguiente proceso: en la primera iteración se selecciona el vértice 1 y se adiciona a X. Luego, en la segunda iteración, se forma el conjunto $M' = \{2, 3, 4\}$ de los vértices con mayor grado complemento (1) y grado (2). Como ninguno de los elementos en M' cumple las condiciones para ser centro, éstos se eliminan de L y dejan como candidatos para la próxima iteración a los vértices 5, 6, 7 y 8. En la siguiente iteración, el conjunto M' sólo está formado por el vértice 5 y como éste tampoco satisface las condiciones de centro es eliminado de L.

Por último, en la cuarta iteración, los vértices restantes en L son eliminados al no cumplir las condiciones exigidas. En este punto la lista L esta vacía y todavía queda el vértice 5 sin agrupar, luego el proceso queda atrapado en un ciclo infinito. Como se mencionó anteriormente, aunque se modifique la condición de parada para evitar el ciclo infinito, el algoritmo todavía dejaría al vértice 5 sin agrupar.

La configuración de vértices del grafo (A) que da lugar a esta situación no es única. En general se cumple que siempre que exista un vértice v, como el del grafo (B) de la figura 2, y éste cumple las condiciones 5 y 6, entonces el algoritmo dejará a v sin agrupar.

$$\forall s_i, 1 \leq i \leq k, |v.Adj| > |s_i.Adj| \tag{5}$$

$$\forall c_i, 1 \leq i \leq k, |c_i.Adj| > |v.Adj| \tag{6}$$

En este grafo, cada s_i es un vértice adyacente de v y cada c_i es el centro de mayor grado adyacente a s_i. En las expresiones 5 y 6, así como en el resto del documento, $v.Adj$ denotará al conjunto de adyacentes de v.

La segunda deficiencia de este algoritmo es que obtiene grupos *redundantes*. En este reporte técnico, se considera que un grupo g_1 es redundante si todos los vértices que pertenecen a g_1 pertenecen también a algún otro grupo diferente de g_1. Esta deficiencia tiene lugar debido a que el algoritmo, para poder ser independiente del orden, permite que más de un vértice que cumpla las condiciones requeridas para ser centro sea considerado

como tal. Como se puede apreciar en el grafo (C) de la figura 2, los grupos formados por los vértices 2 y 3 no deberían existir a la vez, puesto que sólo se necesita uno de ellos para agrupar al vértice 4. Esta deficiencia provoca que aumente la cantidad de grupos producidos.

Una versión diferente de este algoritmo fue presentada por Gil *et al.* en [22] con el objetivo de ser utilizada en un algoritmo paralelo. Esta nueva versión eliminó en el proceso de agrupamiento la verificación de las condiciones de centro y sólo verifica que el elemento a insertar en X no forme un grupo ya existente. Como resultado se mantiene la independencia del orden, la complejidad computacional y se resuelve la primera deficiencia del Extended Star. No obstante a lo anterior, dicha versión produce grupos redundantes y grupos ilógicos.

Algoritmo Generalized Star

El algoritmo Generalized Star fue propuesto por Pérez y Medina [63] y se apoya en el plantamiento de Aslam [4] de que, el cubrimiento de G_β a través de sub-grafos en forma de estrella permite obtener grupos con una semejanza relativamente alta entre los documentos que lo componen. Utilizando este planteamiento y definiendo un nuevo tipo de sub-grafo en forma de estrella, este algoritmo a partir del cubrimiento de G_β con sub-grafos de este tipo, construye un conjunto de grupos que pueden ser solapados.

A diferencia de Star, donde sólo se utiliza el grado de los vértices para definir el sub-grafo, Generalized Star define un grupo de conjuntos para cada uno de los vértices del grafo, y apoyados en estos conjuntos, define lo que es un *sub-grafo en forma de estrella generalizada* y posteriormente, una heurística que define cómo será el agrupamiento que se obtenga.

Los conjuntos de *Satélites Débiles* (WeakSats) y *Satélites Potenciales* (PotSats) de un vértice v, se definen por las siguientes expresiones:

$$v.WeakSats = \{s \in v.Adj \mid |v.Adj| \geq |s.Adj|\},$$

$$v.PotSats = \{s \in v.Adj \mid |v.WeakSats| \geq |s.WeakSats|\}.$$

El *grado WeakSats y PotSats* de un vértice v, se define como la cardinalidad de los conjuntos de satélites débiles y potenciales de v respectivamente.

Teniendo en cuenta las definiciones anteriores, un *sub-grafo en forma de estrella generalizada* (sub-grafo EG) se define como un sub-grafo de $m+1$ vértices, en el cual existe un vértice c denominado "centro" y m vértices llamados "satélites", cumpliéndose que: (i) existe una arista entre cada satélite y el centro, (ii) el centro satisface la expresión 7.

$$\forall s \in c.PotSats, |c.PotSats| \geq |s.PotSats| \tag{7}$$

El pseudo-código del algoritmo GStar puede observarse en el Algoritmo 14.

Algoritmo 14: Generalized Star

Input: $D := \{d_1, d_2, \ldots, d_n\}$ - document collection,
β - similarity threshold
Output: SC - Set of clusters

```
1  "Build G_β = ⟨V, E_β⟩ from D";
2  forall vertex v ∈ V do
3     v.WeakSats := {s | s ∈ v.Adj ∧ |v.Adj| ≥ |s.Adj|};
4  forall vertex v ∈ V do
5  |  v.PotSats := {s | s ∈ v.Adj ∧ |v.WeakSats| ≥ |s.WeakSats|};
6  |  v.NecSats := v.PotSats;
7  end
8  L := V;
9  X := ∅;
10 while L ≠ ∅ do
11 |  v := arg máx{|v_i.PotSats| , v_i ∈ L};
12 |  Update(d,X,L);
13 end
14 "Sort X in ascending order by PotSats degree";
15 SC := ∅;
16 forall center c ∈ X do
17    if c is redundant then  X := X \ {c};
18    else  SC := SC ∪ {{c} ∪ c.Adj};
```

El procedimiento Update (ver Algoritmo 15) se aplica para marcar como centro al vértice seleccionado (v), eliminarlo de la lista de candidatos y para actualizar las propiedades de los vértices del conjunto NecSats del vértice v.

El conjunto de Satélites Necesarios (NecSats) de un vértice v, es el conjunto de los vértices adyacentes que dependen[7] de v para pertenecer a algún grupo. Este conjunto es sólo necesario durante la selección de vértices centro. Inicialmente, el conjunto $v.NecSats$ se inicializa con todos los vértices contenidos en el conjunto $v.PotSats$, pero puede decrecer en la medida que más vértices resulten cubiertos en el proceso de agrupamiento.

El funcionamiento de este algoritmo, se basa en una heurística que asume como criterio para estimar la densidad de un sub-grafo EG, el grado PotSats de su vértice centro. Como primer paso del algoritmo se construye el grafo G_β que representa a la colección. Posteriormente se construye una lista L con todos los vértices del grafo y utilizando ésta, se realiza un proceso iterativo que selecciona en L el vértice v de mayor grado PotSats, se adiciona éste a X, se actualizan sus vértices adyacentes consecuentemente y se elimina de L. El proceso termina cuando la lista L queda vacía.

Al terminar el proceso de cubrimiento, se ordenan los vértices en X de acuerdo a su grado PotSat y se verifica cada centro con el objetivo de eliminar aquellos *centros redundantes* que pudieron ser seleccionados en el proceso anterior. Para determinar si un

[7] En este punto "dependen" hace referencia a que necesitan que el vértice v sea seleccionado como centro

Procedimiento `Update`

Input: v - Selected center,
C- Set of clusters centers,
L - Set of unprocessed vertices
Output: C, L

```
1  X := X ∪ {v};
2  L := L \ {v};
3  forall s ∈ v.NecSats do
4      s.NecSats := s.NecSats \ {v};
5      if s.NecSats = ∅ then L := L \ {s};
6      forall c ∈ s.Adj \ {v} do
7          c.NecSats := c.NecSats \ {s};
8          if (c.NecSats = ∅) ∧ (c.Adj ∩ X ≠ ∅) then L := L \ {c};
9      end
10 end
```

centro es redundante se utiliza el concepto de conjunto de *Centros Potenciales* de un vértice.

El conjunto de *Centros Potenciales* (PotCenters) de un vértice v, es el conjunto de todos los vértices adyacentes a v que potencialmente pueden formar un sub-grafo EG en el cual esté incluido v. Este conjunto está definido por la siguiente expresión:

$$v.PotCenters = \{c \in v.Adj \mid c.PotSats \neq\}$$

Un centro $c \in X$ se considera *redundante* si cumple las condiciones siguientes:

1. $c.PotCenters \cap X \neq \emptyset$; es decir, c tiene algún vértice centro adyacente.
2. $\forall s \in c.PotSats$, $(s \in X) \vee (s.PotCenters \cap (X \setminus \{c\}) \neq \emptyset)$; es decir, cada satélite potencial de c, es centro o tiene algún centro c' adyacente en X tal que $c' \neq c$.

Este algoritmo tiene una complejidad computacional de $O(n^3)$, no necesita del número de grupos a obtener, no produce grupos ilógicos ni redundantes y no deja elementos sin cubrir, solucionando deficiencias de los algoritmos Star y Extended Star.

Este algoritmo puede construir diferentes agrupamientos para una misma colección puesto que sólo selecciona en cada iteración un único elemento para incluir en X, por lo que es dependiente de los datos. Sin embargo, al permitir que los centros de los sub-grafos sean adyacentes, el algoritmo GStar evita descubrir grupos ilógicos, reduce el número de configuraciones en las que el algoritmo pueda dar soluciones diferentes, además de que garantiza obtener una cantidad de grupos menor que los algoritmos anteriores, reduce la influencia del orden de análisis en la calidad de los grupos y obteniendo agrupamientos con valores de calidad muy cercanos.

No obstante a lo anterior, este algoritmo tiene algunas deficiencias. La primera deficiencia es que el algoritmo elimina grupos densos previamente descubiertos. En este reporte

se considera un grupo g_i como denso, si el vértice centro c_i que lo forma es más denso[8] que todos los otros vértices centro contenidos en el grupo. La razón por la cual tiene lugar esta deficiencia es que el criterio adoptado por GStar para decidir cuándo un centro es redundante, no tiene en cuenta la densidad del centro evaluado.

Es importante señalar que la calidad del agrupamiento puede resultar afectada por esta deficiencia, pues basándose en el criterio de agrupamiento utilizado, el grupo eliminado tiene una semejanza interna relativamente alta. En la figura 3 (A) se puede observar un ejemplo de esta deficiencia.

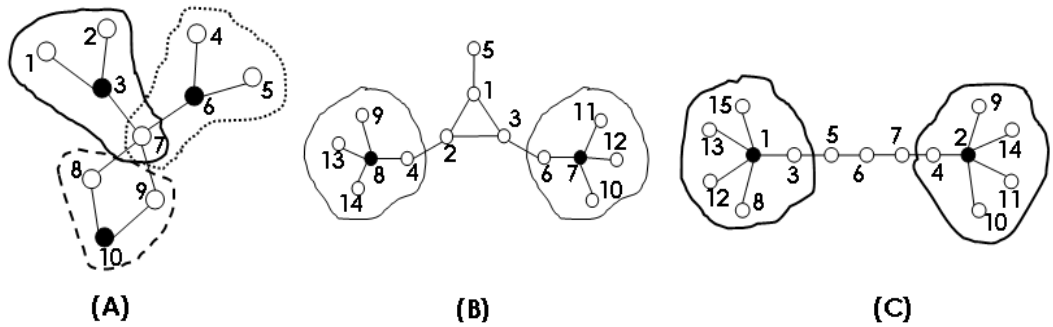

Figura 3. Deficiencias del algoritmo Generalized Star.

La segunda deficiencia de este algoritmo está relacionada con cómo se selecciona el conjunto de centros. Aunque GStar reduce la influencia del orden de los datos en la calidad del agrupamiento resultante al permitir que los centros puedan ser adyacentes, el criterio de agrupamiento utilizado podría mejorarse para reducir aún más el número de configuraciones en las que existe más de un conjunto de grupos como solución.

Considérese la ejecución del algoritmo GStar sobre los grafos (B) y (C) de la figura 3. En la ejecución sobre el grafo (B), una vez que se han insertado en X los vértices 7 y 8, el criterio de agrupamiento considera tres candidatos en la iteración actual, los vértices 1, 2 y 3. De esta forma al terminar el proceso de construcción de X se pueden obtener varios conjuntos posibles: (1) $X = \{7, 8, 3, 1\}$, (2) $X = \{7, 8, 2, 1\}$ o (3) $X = \{7, 8, 1\}$; afortunadamente, el proceso de eliminación de centros redundantes, sin importar cuál fue el conjunto X formado, siempre produciría el mismo agrupamiento final ($X = \{7, 8, 1\}$). Por otra parte, en la ejecución sobre el grafo (C), luego de seleccionar los vértices 1 y 2, los posibles candidatos en la iteración actual son los vértices 5, 6 y 7; luego, en dependencia de la selección, X podría tener diferentes posibles valores; lamentablemente en este caso, el proceso de eliminación de centros redundantes no soluciona este problema.

La última deficiencia de este algoritmo está relacionada con el consumo de memoria. GStar necesita para su funcionamiento el cálculo de varios conjuntos para cada vértice,

[8] El criterio de densidad que utiliza el algoritmo GStar tiene en cuenta solamente el grado PotSats del vértice centro del sub-grafo que forma al grupo

como por ejemplo los adyacentes, WeakSats, PotSats, PotCenters e incluso NecSats, los cuales debe mantener en memoria hasta el final del algoritmo. Debido a lo anterior, este algoritmo pudiera resultar ineficiente con grandes colecciones de documentos.

Algoritmo CStar

El algoritmo CStar[9][18] define un nuevo concepto de sub-grafo llamado *sub-grafo en forma de estrella condensada* y aplica una heurística, a partir de la cual obtiene un cubrimiento de G_β utilizando sub-grafos de este tipo. Este algoritmo obtiene un conjunto de grupos que pueden tener traslape y además, mantiene las bondades de sus predecesores a la vez que reduce algunas de las limitaciones presentes en éstos.

Antes de describir el algoritmo CStar, se necesita introducir algunos conceptos básicos que se utilizan para definir el nuevo concepto de sub-grafo en forma de estrella y el nuevo criterio de agrupamiento.

Sea $G_\beta = \langle V, E_\beta \rangle$ un grafo de β-semejanza, llamaremos *0-Transición* de G_β al grafo dirigido $G_\beta^{(0)} = \langle V, E_\beta^{(0)} \rangle$ que se obtiene adicionando por cada arista no-dirigida $(u,v) \in E_\beta$ las aristas dirigidas (u,v) y (v,u) en $E_\beta^{(0)}$. En la figura 4 se muestra un ejemplo de la *0--Transición* de un grafo de β-semejanza; en ambos grafos los vértices están etiquetados con el valor de su grado saliente.

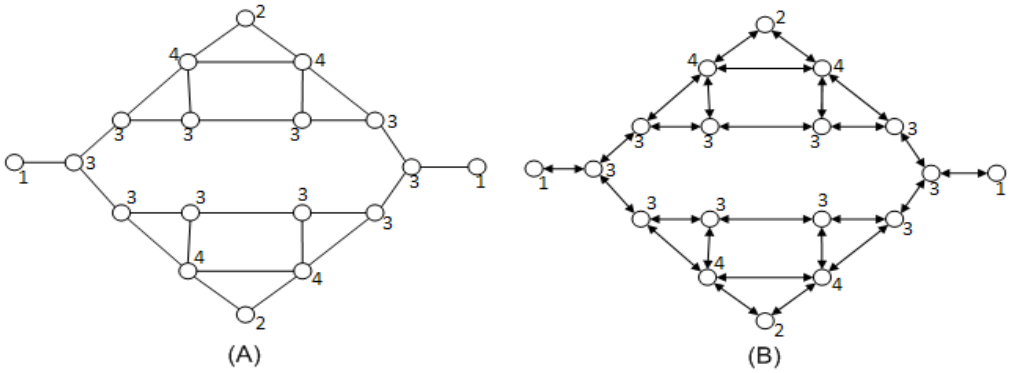

Figura 4. (A) $G_\beta = \langle V, E_\beta \rangle$, $|E_\beta| = 26$, (B) $G_\beta^{(0)} = \langle V, E_\beta^{(0)} \rangle$, $\left|E_\beta^{(0)}\right| = 52$

Sea Γ una función definida como:

$$\Gamma : G = \langle V, E \rangle \longrightarrow \widetilde{G} = \langle V, \widetilde{E} \rangle,$$

donde G y $\widetilde{G}$ son grafos dirigidos. El grafo $\widetilde{G}$ se construye a partir de G eliminando de E las aristas dirigidas (u,v) que no satisfacen:

$$u.out \geq v.out, \tag{8}$$

[9] Este algoritmo fue publicado con el nombre ACONS

donde $v.out$ y $u.out$ son los grados salientes de v y u respectivamente. Al grafo $\widetilde{G}$ se le denomina Γ-*Transición* de G.

En la figura 5 se muestra un ejemplo de la Γ-*Transición* de un grafo de β-semejanza. En este ejemplo, los vértices que perdieron al menos una arista aparecen resaltados en color negro.

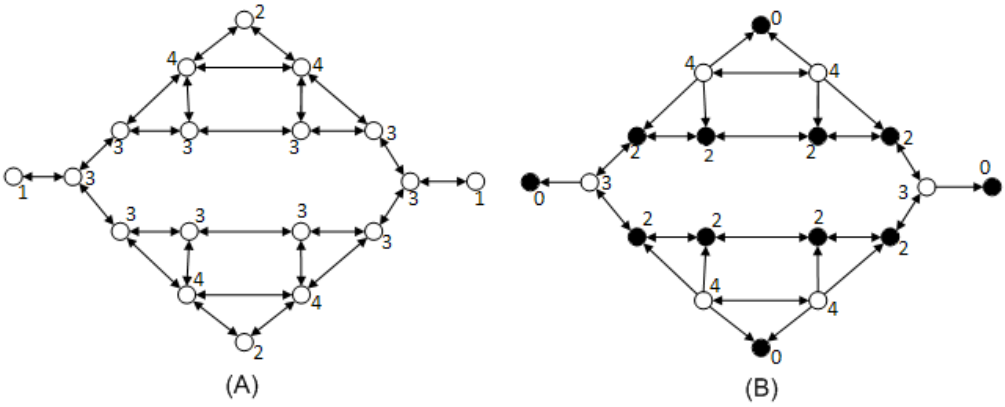

Figura 5. (A) $G = \langle V, E\rangle$, (B) $\widetilde{G}$, la Γ-*Transición* de G

Teniendo en cuenta los conceptos previos, se construye una secuencia de grafos $\{G_\beta^{(0)}, G_\beta^{(1)}, G_\beta^{(2)}, \ldots, G_\beta^{(n)}, \ldots\}$, de forma tal que para todo $i > 0$, $G_\beta^{(i)} = \langle V, E_\beta^{(i)}\rangle$ es la Γ-*Transición* de $G_\beta^{(i-1)}$. Esta secuencia se llama *secuencia de* Γ-*Transiciones de* G_β y cumple que la sucesión de números enteros no-negativos $\{e_n\}_{n=0}^{\infty}$, donde $e_n = \left|E_\beta^{(n)}\right|$, es monótona decreciente acotada inferiormente por 0.

En la figura 6 se muestran los grafos $G_\beta^{(1)}, G_\beta^{(2)}, G_\beta^{(3)}, G_\beta^{(4)}$ de la secuencia de Γ-Transiciones del grafo G_β de la figura 4 (A).

Como todos los grafos $G_\beta^{(i)}$ de la secuencia de Γ-Transiciones de G_β comparten el mismo conjunto de vértices, se referenciará al grado saliente del vértice v en el grafo $G_\beta^{(i)}$ como $v.out[i]$. En [61] se presenta un conjunto de teoremas y corolarios que permiten asegurar que el proceso de construcción de la secuencia de Γ-Transiciones de G_β es finito y no genera vértices aislados.

A continuación se introducen algunos conceptos que permiten definir el concepto de sub-grafo en forma de Estrella Condensada.

Definición 2.1. *Sea* h *el primer entero tal que* $G_\beta^{(h+1)} = G_\beta^{(h)}$, *a éste entero* h *se llamará* cota *del grafo* G_β *y a* $G_\beta^{(h)}$ última transición *del grafo* G_β

Sea $G_\beta = \langle V, E_\beta\rangle$ un grafo de β-semejanza, h la cota de G_β y $u, v \in V$; u es un r-*satélite* de v, con $0 \leq r \leq h$, si se cumple que la arista dirigida $(v,u) \in E_\beta^{(r)}$. El conjunto de todos los r-satélites de v se denota por $v.Sats[r]$ y se define por la siguiente expresión:

$$v.Sats[r] = \{u \in V \mid u \text{ es un } r\text{-satélite de } v\}$$

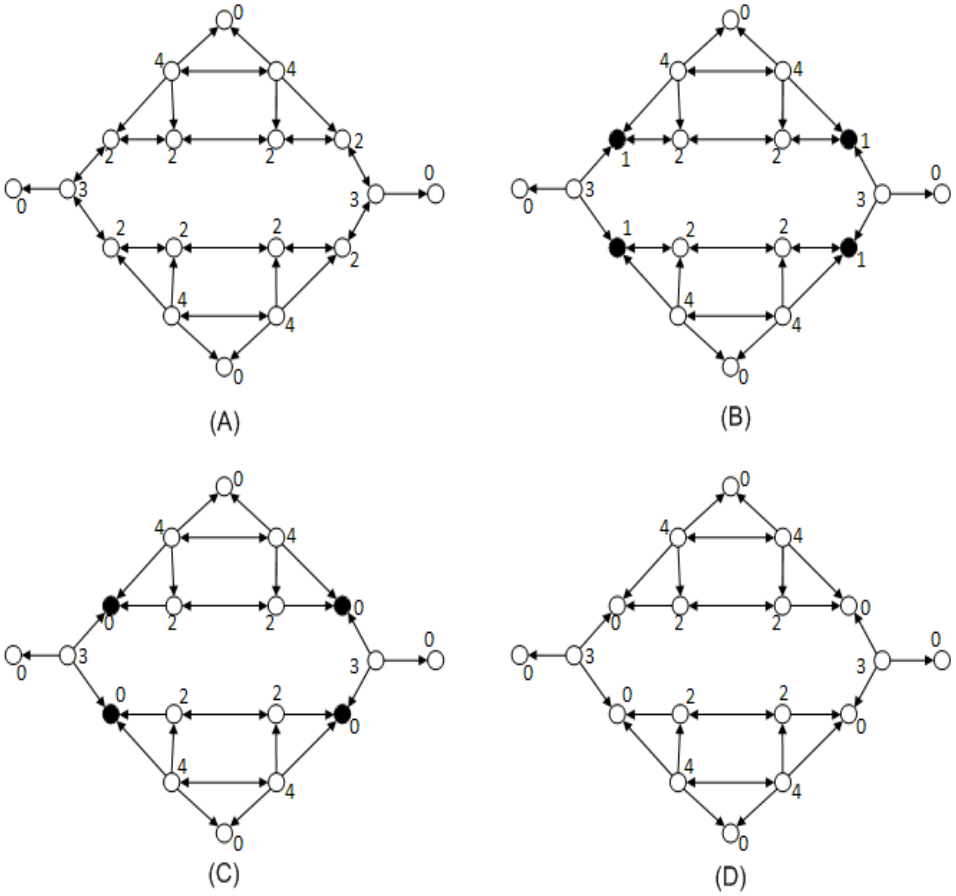

Figura 6. (A) $G_\beta^{(1)}$, $\left|E_\beta^{(1)}\right| = 38$, (B) $G_\beta^{(2)}$, $\left|E_\beta^{(2)}\right| = 34$, (C) $G_\beta^{(3)}$, $\left|E_\beta^{(3)}\right| = 30$, (D) $G_\beta^{(4)}$, $\left|E_\beta^{(4)}\right| = 30$

En la figura 7 se muestra un ejemplo de 4-satélites de un vértice; en esta figura, el grafo (A) es la 4^{ta} transición de un grafo de β-semejanza y el grafo (B) es el mismo grafo (A) pero en el cual los vértices se han etiquetado con letras en lugar de con su grado saliente. En el grafo (B) los 4-satélites de los vértices d y n aparecen resaltados en color gris.

Sea $G_\beta = \langle V, E_\beta \rangle$ un grafo de β-semejanza, h la cota de G_β, un *sub-grafo en forma de Estrella Condensada* (sub-grafo EC) es un sub-grafo de $m+1$ vértices, en el cual existe un vértice c denominado "centro" y m vértices llamados "satélites", cumpliéndose que: (i) $c.out[h] > 0$ y (ii) existe una arista entre cada satélite y el centro c en G_β. Como caso particular de la definición anterior, cada vértice aislado $u \in V$ se considera un sub-grafo EC degenerado de un sólo vértice, el vértice centro.

En la figura 8, (A) es la última transición de un grafo de β-semejanza G_β y (B) muestra un ejemplo de dos posibles sub-grafos EC. En el grafo (B), aquellos vértices que aparecen en color negro corresponden con los centros de los sub-grafos EC.

Teniendo en cuenta los conceptos previos, se puede enunciar que la idea principal del algoritmo CStar es definir un criterio que establezca cuándo un sub-grafo EC es más denso

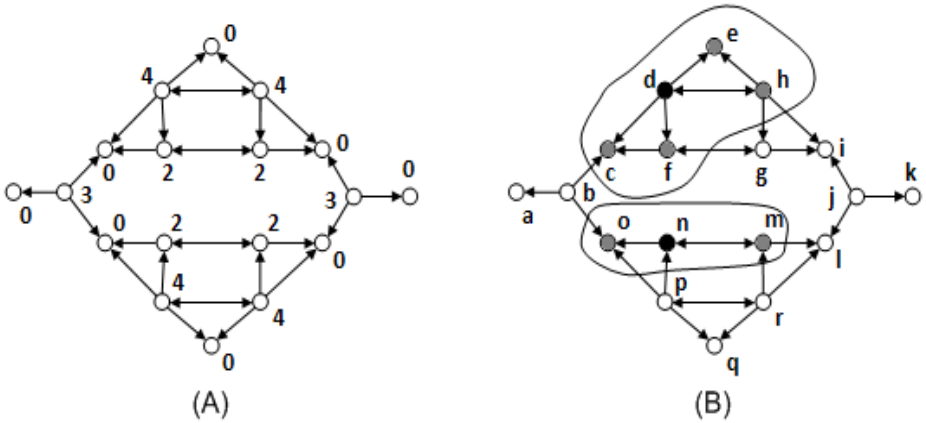

Figura 7. Ejemplo de los 4-satélites de un vértice

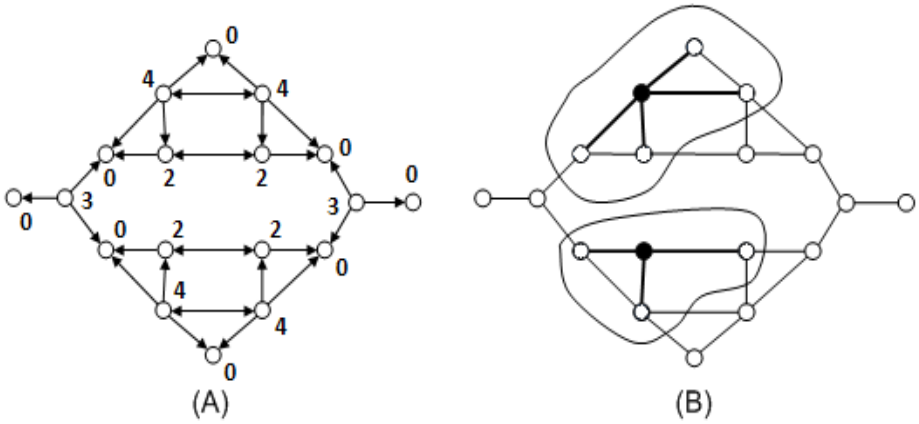

Figura 8. Ejemplo de dos posibles sub-grafos EC

que otro y a partir de éste, realizar un cubrimiento de G_β utilizando los sub-grafos EC más densos y posteriormente aplicar un proceso de filtrado que permita reducir la cantidad de éstos.

El problema de encontrar un cubrimiento de G_β puede transformarse en el problema de encontrar el conjunto X de vértices, tal que cada cada vértice $c_i \in X$ determine un sub-grafo EC. Un aspecto importante para solucionar este problema es reducir el número de vértices candidatos que pueden ser seleccionado como centro.

Sea $G_\beta^{(h)}$ la última transición de $G_\beta = \langle V, E_\beta \rangle$, v un vértice de V y el conjunto de vértices $F = \{x \mid x \in v.Adj \cup \{v\}\}$; el conjunto de vértices adyacentes a v que mayor posibilidad tienen de agrupar a éste se denota por $v.Electees$ y está definido por la siguiente expresión:

$$v.Electees = \{x \mid x \in F \wedge x.out[h] = max\{w.out[h] \mid w \in F\}\}$$

Sea $G_\beta^{(h)}$ la última transición de $G_\beta = \langle V, E_\beta \rangle$ y $v \in V$ un vértice no aislado. El *grado de votación* de v se denota por $v.vd$ y se define por la siguiente expresión:

$$v.vd = |\{u \mid u \in v.Adj \wedge v \in u.Electees\}|$$

El grado de votación de un vértice v permite estimar cuántos de sus h-satélites necesitan de él para pertenecer a algún sub-grafo EC; es decir, la mayor cantidad de vértices no agrupados por otros vértices con grado de votación superior al de v que puede agrupar v si se adiciona al conjunto X. El concepto de grado de votación de un vértice guarda relación con el PageRank de las páginas de Google [40], lo que a diferencia de éste, la relevancia de cada nodo en el grafo se estima a través de sus aristas salientes.

La lista de candidatos L que considera el algoritmo CStar está compuesta por todos los vértices $v \in V$ tal que $v.vd > 0$; de esta forma, CStar evita seleccionar como centros a vértices que forman grupos que a *priori* son redundantes al no agrupar a ningún vértice no cubierto.

Teniendo en cuenta el grafo que se ha desarrollado en ejemplos anteriores, en la figura 9 se muestra un ejemplo del cálculo del grado de votación de los vértices en un grafo de β-semejanza G_β. En esta figura, los grafos (A) y (B) representan a $G_\beta^{(h)}$ y a G_β respectivamente; en (A) los vértices están etiquetados con su grado saliente y en (B) con su grado de votación. Note que los vértices que en el grafo (B) tienen un grado de votación diferente de cero aparecen resaltados con color negro.

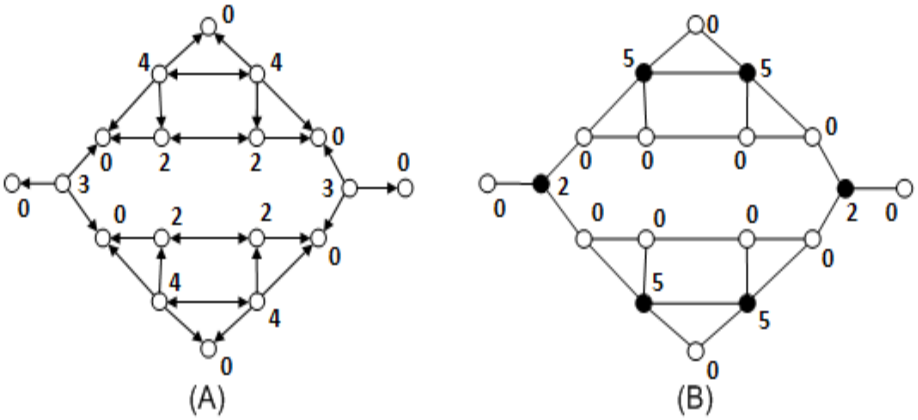

Figura 9. Ejemplo del cálculo del grado de votación

A partir de esta lista, CStar selecciona iterativamente un vértice v y lo inserta en X si v forma el sub-grafo EC más denso. Existen varios criterios para determinar la densidad de un sub-grafo, como por ejemplo, los utilizados por los algoritmos Star y Generalized Star. El algoritmo CStar considera, para determinar entre dos sub-grafos el más denso, el grado de votación de sus vértices centro. Por lo tanto, seleccionar en cada iteración el vértice centro de mayor grado de votación, permite a CStar formar el grupo con más vértices no cubiertos y reducir el número de vértices necesarios para cubrir a G_β.

El pseudo-código de CStar se muestra en el Algoritmo 16.

Algoritmo 16: CStar

Input: $D := \{d_1, d_2, \ldots, d_N\}$ - document collection
β - similarity threshold
Output: SC - set of clusters

`// Phase 1 - Building the candidate list L`

"Build $G_\beta = \langle V, E_\beta \rangle$ from D";
$G_\beta^{(0)} :=$ `BuildTransition0(`G_β`)`;
$G_\beta^{(h)} :=$ `FindLastTransition(`$G_\beta^{(0)}$`)`;
`CalculateVotingDegree(`V`)`;
$L := \{v \in V \mid v.vd > 0\}$;

`// Phase 2 - Computing centers list` X `and uncertain centers list` U

$X := \{v \in V \mid v.Adj = \emptyset\}$;
$U := \emptyset$;
while $L \neq \emptyset$ **do**
  $v := \arg max_x\{x.vd \mid x \in L\}$; `// Only one vertex is selected`
  if $v.Adj \cap X = \emptyset$ **then** $X := X \cup \{v\}$
  else
    $F = \{u \in v.Sats[h] \mid u.Adj \cap X = \emptyset\}$;
    if $F \neq \emptyset$ **then**
      if $\exists f \in F$, $v.vd > f.vd$ **then** $X := X \cup \{v\}$
      else $U := U \cup \{v\}$;
    end
  end
  $L := L \setminus \{v\}$;
end

`// Phase 3 - Selecting new centers from uncertainty list` U

forall *vertex* $v \in U$ **do**
  if $\exists u \in v.Sats[h]$, $u.Adj \cap X = \emptyset$ **then** $X := X \cup \{v\}$;
end

`// Phase 4 - Removing redundant centers from` X

"Sort X in ascending order by voting degree";
$SC := \emptyset$;
forall *center* $c \in X$ **do**
  if c *is redundant* **then** $X := X \setminus \{c\}$
  else $SC := SC \cup \{\{c\} \cup c.Adj\}$;
end

Las funciones $BuildTransition0$ y $FindLastTransition$ se utilizan para construir $G_\beta^{(0)}$ y encontrar $G_\beta^{(h)}$ respectivamente. La función $CalculateVotingDegree$ calcula el grado de votación para cada vértice de $G_\beta^{(h)}$.

El algoritmo CStar está compuesto por 4 fases o etapas: (1) construcción de la lista de vértices candidatos L, (2) construcción del conjunto inicial de centros X y la lista de vértices *dudosos* U, (3) selección de nuevos vértices centro en U y adición de éstos a X, (4) eliminación del conjunto X de todo vértice que forme un grupo *redundante*. A continuación se describe en detalle el funcionamiento del algoritmo.

La primera etapa obtiene un conjunto reducido de vértices que pueden formar sub-grafo EC y que tienen las mayores posibilidades de ser seleccionados como centro. Como la lista de candidatos L está formada por vértices con grado de votación no nulo, los vértices que no pertenezcan a L son aislados (forman un sub-grafo EC degenerado y por tanto son centros) o tienen algún vértice adyacente en L. Luego, en caso de que se inserten en X todos los vértices de L, se garantiza que no queden vértices sin agrupar.

Al comienzo de la etapa 2, se inicializa el conjunto X con todos aquellos vértices aislados de G_β. Posteriormente, la lista L es procesada iterativamente en orden decreciente respecto al grado de votación; este orden garantiza que el vértice $v \in L$ seleccionado en cada iteración sea el que forma el sub-grafo más denso de los candidatos en esa iteración y por lo tanto, se evita descubrir grupos ilógicos. El vértice v es procesado considerando las siguientes situaciones:

1. Si v no está agrupado, entonces v es centro y se inserta en X. Al priorizar, en el proceso de selección a los vértices no agrupados, se permite realizar un cubrimiento más rápido de G_β y se evita forzar el cubrimiento entre los grupos.
2. Si v está agrupado, entonces la decisión de seleccionar a v como centro depende de los vértices no agrupados de su conjunto de h-satélites. Sea F el conjunto de vértices no agrupados del conjunto v.Sats[h]. Si $F = \emptyset$, v no puede ser seleccionado ya que forma un grupo redundante. En otro caso, como v es necesario todavía para agrupar a algún vértice, se procede considerando las condiciones siguientes:
   a) Si v tiene al menos un vértice $f \in F$ tal que $v.vd > f.vd$, entonces v es centro y se inserta en X. Esta condición permite agrupar a f en un sub-grafo más denso que el que f podría formar si se selecciona en alguna iteración posterior. Esta forma de proceder evita construir grupos ilógicos.
   b) En otro caso, v se clasifica como dudoso y es insertado en la lista U posponiendo su posible selección como centro. Esta condición permite explorar otras posibles selecciones de vértices adyacentes a v que tengan su mismo valor de densidad y que puedan agrupar a éste y a algún adyacente menos denso. Nótese que, aunque este paso puede hacer que el agrupamiento difiera de una ejecución a otra, se garantiza que los sub-grafos formados tengan la misma densidad, pues en otro caso se hubiese seleccionado v por la condición a.

Después de este proceso, el vértice v es eliminado de la lista L garantizando de esta forma que se alcance la condición de parada de la etapa 2. Durante la etapa 3, todos los vértices de U son verificados en el mismo orden en que fueron insertados, garantizando de esta forma procesar siempre al vértice de la lista U que forma el sub-grafo EC más denso de todos los posibles. Para cada uno de estos vértices v, se verifica si tienen algún h-satélite no agrupado y si es el caso, se selecciona como centro y se inserta en X.

Es importante resaltar que una vez concluidas las etapas 2 y 3, los vértices que no fueron seleccionados como centro en dichas etapas cumplen que: (i) ya están agrupados; en otro caso hubieran sido seleccionados como centro por la condición 1 de la etapa 2 y (ii) todos sus h-satélites ya están agrupados; en otro caso hubieran sido seleccionado como centro. Luego, queda garantizado que cada vértice de G_β pertenece al menos a un grupo.

Finalmente la etapa 4 ordena los vértices en X ascendentemente de acuerdo a su grado de votación y elimina de X aquellos que sean redundantes. Al finalizar, el agrupamiento se forma con los vértices que quedaron en X.

Un vértice centro $c \in X$ se considera *redundante* si satisface las siguientes condiciones:

1. $\exists d \in c.Adj \cap X, d.vd > c.vd$; es decir, el vértice c tiene adyacente a él al menos un vértice centro con mayor grado de votación.
2. $\forall s \in c.Sats[h], s \in X \vee (s.Adj \cap (X \setminus \{c\})) \neq \emptyset$; es decir, cada h-satélite s de c es centro o tiene algún vértice centro diferente de c adyacente a él.

La complejidad computacional del algoritmo CStar es $O(h \cdot n^2)$, donde h es la cota de G_β. En [61] se presenta un conjunto de teoremas y corolarios que determinan el valor máximo que puede alcanzar h.

Algoritmo CStar+

El algoritmo CStar+[62], es una variante del CStar y el mismo realiza un cubrimiento sobre las componentes conexas de G_β utilizando sub-grafos EC. Dado que se tiene el conjunto $CP = \{CP_1, CP_2, \ldots, CP_k\}$ de componentes conexas de G_β, CStar+ transforma el problema de determinar un agrupamiento de G_β a través de sub-grafos EC en el problema de realizar un cubrimiento utilizando sub-grafos EC de cada componente conexa CP_i, $i = 1...k$ y este último, en el problema de encontrar el conjunto X_i de vértices centro que forman dichos sub-grafos. A continuación se describe el algoritmo CStar+.

Es importante tener en cuenta que aunque obtener un cubrimiento de estas componentes a través de sub-grafos EC reduce el encadenamiento, también podría afectar la calidad del agrupamiento si dicha componente tiene un alto grado de conexión entre sus vértices, pues se estaría dividiendo en sub-grupos un grupo que ya es altamente cohesionado. Un ejemplo de grafo conexo altamente conectado es el *cuasi-clique*.

Sea $G = \langle V, E \rangle$ un grafo conexo, G se considera un *cuasi-clique* si se cumple que:

$$\forall v \in V, |v.Adj| \geq \alpha \cdot (n-1),$$

donde $n = |V|$ y α es un número real $0 < \alpha \leq 1$. Este tipo de grafo puede considerarse como un clique en el cual cada vértice ha perdido un número de aristas. Existen trabajos que han utilizado este tipo de grafo para el análisis de relaciones entre entidades [49] y en el análisis de genes y proteínas [47].

Es importante mencionar que α sólo representa, de forma similar al umbral β, un valor de confianza definido por el usuario para la aceptación o no de una componente conexa como grupo cohesionado. El valor de α está muy relacionado con el valor de β ya que al ser menor el valor de éste último existen más enlaces entre los vértices de G_β y aumenta la probabilidad de que las componentes conexas de G_β sean cuasi-cliques; por lo tanto, el valor de α debe ser mayor para detectar grupos que realmente son altamente conectados.

No obstante a lo anterior, según Pei [52], para lograr detectar sub-grafos realmente cohesionados (de menor diámetro) los valores de α deben encontrarse en los intervalos $(\frac{n-2}{n-1};1]$ ó $[0.50;\frac{n-2}{n-1}]$, pues en éstos se obtienen sub-grafos con diámetros menor que 2 y por lo tanto altamente cohesionados. En los intervalos anteriores n representa la cantidad de vértices del sub-grafo.

Como se explicó en la sección anterior, uno de los pasos del algoritmo CStar es la construcción de una lista de candidatos L a partir de $G_\beta^{(h)}$. Considerando que el grafo G_β se divide en componentes conexas $CP_1, CP_2, \ldots, CP_k$ sobre las cuales se realiza un cubrimiento a través de sub-grafos EC, puede ocurrir que existan componentes en las cuales no sea necesario construir la secuencia de transiciones; estas componentes conexas son los *k-plex*.

Sea $G = \langle V, E \rangle$ un grafo conexo, G se considera un *k-plex* si se cumple que:

$$\forall v \in V, v.Adj = (n - k),$$

donde $n = |V|$. Detectar este tipo de componente permite evitaría a CStar la construcción de la 0-Transición y de la última transicion de G, de esta forma se reduce el tiempo necesario por el algoritmo para cubrir G.

La idea del algoritmo CStar+ consiste en dada una colección de documentos, construir su grafo de semejanza G_β y el conjunto de componentes conexas CP que forman a dicho grafo. Posteriormente se procesa cada componente conexa CP_i y se procede según las condiciones siguientes:

1. Si CP_i es un cuasi-clique entonces es un grupo del agrupamiento resultante.
2. Si CP_i es un k-plex entonces no es necesario realizar las transiciones sobre CP_i y se realiza un cubrimiento de ésta utilizando el algoritmo CStar considerando la lista L compuesta por todos los vértices de la componente.
3. En otro caso utilizar al algoritmo CStar para obtener el cubrimiento de CP_i.

Al terminar el proceso anterior, el agrupamiento resultante se forma por la unión de los grupos detectados en cada componente conexa. El pseudo-código del algoritmo CStar+ se muestra en el Algoritmo 17.

Algoritmo 17: CStar+

Input: $D := \{d_1, d_2, \ldots, d_N\}$ - document collection
β - similarity threshold
α - connection threshold
Output: SC - set of clusters

"Constructs G_β and the connected component set $CP = \{CP_1, CP_2, \ldots, CP_N\}$ from D";
$SC := \emptyset$;
foreach *component* $\widetilde{G}_\beta = \left\langle V', \widetilde{E}_\beta \right\rangle \in CP$ **do**
  if $\widetilde{G}_\beta$ *is a quasi-clique* **then** $SC := SC \cup V'$;
  else
    `// Phase 1`
    $\widetilde{G}_\beta^{(0)} :=$ `BuildTransition0(`$\widetilde{G}_\beta$`)`;
    $\widetilde{G}_\beta^{(h)} :=$ `FindLastTransition(`$\widetilde{G}_\beta^{(0)}$`)`;
    `CalculateVotingDegree(`V'`)`;
    $L := \{v \in V' \mid v.vd > 0\}$;
    `// Phase 2`
    $X := \emptyset$;
    $U := \emptyset$;
    while $L \neq \emptyset$ **do**
      $v := \arg max_x\{x.vd \mid x \in L\}$; `// Only one vertex is selected`
      if $v.Adj \cap X = \emptyset$ **then** $X := X \cup \{v\}$
      else
        $F = \{u \in v.Sats[h] \mid u.Adj \cap X = \emptyset\}$;
        if $F \neq \emptyset$ **then**
          if $\exists f \in F$, $v.vd > f.vd$ **then** $X := X \cup \{v\}$
          else $U := U \cup \{v\}$;
        end
      end
      $L := L \setminus \{v\}$;
    end
    `// Phase 3`
    forall *vertex* $v \in U$ **do**
      if $\exists u \in v.Sats[h]$, $u.Adj \cap X = \emptyset$ **then** $X := X \cup \{v\}$;
    end
    `// Phase 4`
    "Sort X in ascending order by voting-degree"; $SC := \emptyset$;
    forall *center* $c \in X$ **do**
      if c *is redundant* **then** $X := X \setminus \{c\}$
      else $SC := SC \cup \{\{c\} \cup c.Adj\}$;
    end
  end
end

La complejidad computacional de este algoritmo es $O(h \cdot n^2)$ puesto que el paso de calcular las componentes conexas es $O(n^2)$ y los otros pasos se reducen a aplicar el algoritmo CStar en el grafo que determina cada componente conexa. Es importante resaltar que, al construir G_β y sus componentes conexas, para cada componente conexa $\widetilde{G}_\beta = \left\langle V', \widetilde{E}_\beta \right\rangle$ se almacena el menor y mayor grado de un vértice $v \in V'$. Luego, en el proceso de agrupamiento se utilizan estos valores para determinar en $O(1)$ el tipo de componente de $\widetilde{G}_\beta$.

Es importante resaltar que, aunque el valor del parámetro (α) sólo representa un umbral para lo que el usuario considera o no un grupo cohesionado, el algoritmo CStar+ es sensible al valor asignado a α, pues valores pequeños podrían formar grupos con poca cohesión interna ya que se asumirían como grupos sub-grafos poco conectados.

2.11. Algoritmos híbridos

Este tipo de algoritmos, como su nombre lo indica, utiliza una combinación de algoritmos de distintos tipos para encontrar la estructuración de la colección. Ejemplos de estos algoritmos son el Scatter/Gather [15], el ICT [45] y el UMASS [13].

El Scatter/Gather es un algoritmo que combina las ventajas de un algoritmo jerárquico aglomerativo y uno basado en optimización. Scatter/Gather representa los documentos mediante el modelo de Espacio Vectorial y utiliza la medida del coseno para calcular la semejanza entre los documentos. Como primer paso Scatter/Gather aplica un algoritmo aglomerativo para encontrar los k centroides y una vez encontrados aplica un algoritmo de optimización para refinar los grupos.

El algoritmo aglomerativo que se aplica es el Average-link (ver sección 2.3) y puede aplicarse siguiendo una estrategia *Buckshot* o *Fractionation*. Luego de que se tienen determinados los k centroides se aplica un proceso de refinamiento de la partición en el cual puede utilizarce: (i) una variante del algoritmo k-means, (ii) el algoritmo Split-Merge; el cual iterativamente divide los grupos con menor semejanza intracluster y une aquellos grupos muy semejantes, o (iii) una combinación de ambos. Ver [15] para más información.

El pseudo-código del algoritmo Scatter/Gather puede observarse en el Algoritmo 18. En este pseudo-código se utilizó para la fase de refinamiento el algoritmo k-means y para la formación de los grupos iniciales el algoritmo UPGMA[10].

Los algoritmos ICT y UMASS han sido utilizados para la detección jerárquica de tópicos en colecciones de noticias [74]. Como paso previo para el agrupamiento estos algoritmos utilizan la información temporal de las noticias para ordenarlas cronológicamente. En el algoritmo ICT se aplica un algoritmo aglomerativo para construir la jerarquía y un algoritmo de optimización o de una sola pasada para refinar cada nivel de ésta. El algoritmo aglomerativo utilizado fue el Single-link utilizando representantes para los grupos y definiendo un valor de umbral de semejanza para cada nivel. Para refinar las particiones de cada nivel se utilizó el Single-pass o el K-means.

[10] Una descripción de este método puede verse en la sección 2.3

Algoritmo 18: Scatter/Gather

Input: $D := \{d_1, d_2, \ldots, d_n\}$ - document collection,
β - similarity threshold
Output: SC - Set of clusters

```
1  SC := UPGMA(V,β);
2  forall C_i ∈ SC do
3  |   C_i.Center := Average(C_i);
4  |   C_i = ∅;
5  end
   // adjusting the clusters using the k-means approach
6  repeat
   |   // forming the set of clusters
7  |   forall d_j ∈ V do
8  |   |   i := arg_i min{F(d_j, C_i.Center), C_i ∈ SC};
9  |   |   C_i := C_i ∪ d_j;
10 |   end
11 |   cond := Evaluate(a,b) ;                    // Evaluating the stop-conditions
   |   // updating cluster's representative
12 |   forall C_i ∈ SC do
13 |   |   C_i.Center := Average(C_i);
14 |   |   C_i := ∅;
15 |   end
16 until cond ≠ true ;
```

El algoritmo UMASS, una vez ordenadas las noticias cronológicamente, utiliza el algoritmo 1-NN [74] para agruparlas. Para realizar esto, al procesar cada noticia q se calcula la semejanza de ésta con las k noticias anteriores, seleccionándose aquellas cuya semejanza excede cierto umbral β. Posteriormente, la noticia q se le asigna al grupo al que pertenece la noticia más semejante a q; si no existe tal grupo, se forma un nuevo grupo unitario que contiene a q. Luego del paso anterior, los grupos obtenidos se dividen en sub-conjuntos y sobre cada uno de ellos se ejecuta un algoritmo aglomerativo para reducir el número de grupos determinados. Como último paso, los grupos resultantes de cada sub-conjunto son agrupados con el algoritmo aglomerativo para formar la jerarquía.

Estos algoritmos son dependientes del orden. En Scatter/Gather además la calidad del agrupamiento puede depender de la muestra seleccionada si se utiliza una estrategia *Buckshot*. Los grupos formados por el ICT pueden presentar encadenamientos al utilizarse el Single-link. Por último cabe mencionar que los tres algoritmos necesitan prefijar valores de parámetros para ejecutarse.

3. Evaluación experimental

Considerando que en el contexto del agrupamiento de documentos es deseable que se tenga en cuenta la pertenencia de un documento a más de un grupo, en esta sección se presenta una serie de experimentos en los que se evalúa la calidad del agrupamiento obtenido por varios algoritmos expuestos en la sección 2.

Los algoritmos seleccionados los podemos agrupar en dos grupos: (a) Star, Extended Star, Generalized Star, Fuertemente Compacto Incremental, FTC[11], CStar y CStar+, que permiten la obtención de cubrimientos y (b) los algoritmos clásicos Single-link y Average--link, que obtienen grupos disjuntos pero que son frecuentemente utilizados en el agrupamiento de documentos. Todos los algoritmos fueron implementados en $C++$ al no estar disponibles los códigos originales y además fueron incluidos en una plataforma desarrollada en .Net[19] que permite la evaluación de éstos en diferentes repositorios y considerando varias medidas de evaluación.

Teniendo en cuenta las bondades de los algoritmos CStar y CStar+ expuestas en [61], los experimentos mostrados en esta sección evaluarán el comportamiento de los algoritmos CStar y CStar+ respecto al comportamiento del resto de los algoritmos anteriormente mencionados.

3.1. Descripción de las colecciones

Para evaluar la calidad del agrupamiento obtenido por los algoritmos, se seleccionaron dos colecciones de documentos ampliamente utilizadas en evaluaciones de algoritmos de agrupamiento de documentos: TREC-5 [75] y *Reuters*-21578 [43]. Estas colecciones son variadas en cuanto al tamaño de los documentos, número de clases, tamaño de las clases, dimensionalidad, etc.

La colección TREC-5 contiene 695 artículos periodísticos en castellano publicados por la agencia AFP en el año 1994 y que han sido clasificados en 25 tópicos o clases. La colección *Reuters*-21578 contiene 21578 artículos periodísticos en inglés de la agencia Reuters publicados en el año 1987 y que están organizados en 120 clases. De esta última colección se seleccionaron 10377 documentos que están clasificados en uno o más tó picos y que tienen al menos un término.

En lo adelante se referirá a las colecciones TREC-5 y *Reuters*-21578 como AFP y *Reuters* respectivamente. Algunas de las características de estas colecciones se presentan en la tabla 3.

En esta tabla, "N. Doc" es el número de documentos de la colección, "N. Clases" representa la cantidad de clases en las que se agrupan los documentos de la colección[12], "Solapamiento" hace referencia a la cantidad de documentos que están asignados a más de

[11] Aquí se consideró una versión del algoritmo FTC que permite obtener cubrimiento y que fuera propuesta por el autor de dicho algoritmo

[12] Estas clases fueron detectadas manualmente por los expertos

Tabla 3. Características de las colecciones de documentos AFP y *Reuters*

Colección	N. Doc	Solapamiento	N. Clases	N. térm	Prom. térm	Idioma
AFP	695	16	25	11839	126	Español
Reuters	10377	1722	119	21218	43	Inglés

una clase; finalmente, "N. térm" y "Prom. térm" son el número de términos presentes en la colección luego de lematizar sus documentos y el promedio de términos por documentos respectivamente.

3.2. Representación de los documentos

Los documentos de ambas colecciones fueron representados utilizando el Modelo de Espacio Vectorial. El peso de cada término en el vector documento se determinó a partir del valor de la frecuencia del término normalizada por la frecuencia máxima de los términos en el documento [23].

Los términos en los documentos representan la forma reducida (lema) de las palabras que aparecen en el documento, por ejemplo, todas las conjugaciones de un verbo se representaron por su infinitivo, el plural de la palabra por su forma singular, etc. En este conjunto de términos no se consideraron las palabras vacías presentes en el documento. Algunas de las palabras vacías no consideradas como términos son: artículos, preposiciones, adverbios, conjunciones, etc.

Para obtener los lemas de los términos de ambas colecciones se utilizó el etiquetador léxico-morfológico *TreeTagger*[13], desarrollado en el Instituto de Lingüística Computacional de la Universidad de Stuttgart. La función fundamental del TreeTagger es analizar y desambiguar las categorías léxicas de las palabras. Para ello se basa en los modelos de Markov de orden k y en los árboles de decisión con objeto de obtener estimaciones más fiables.

Para determinar la semejanza entre los documentos se utilizó la medida del coseno [7].

3.3. Medidas de evaluación de la calidad

Existen variadas medidas para estimar la calidad de los agrupamientos de documentos obtenidos por un algoritmo. Estas medidas se clasifican en medidas *internas* y *externas* atendiendo a si utilizan o no alguna información externa al agrupamiento. A continuación, se describen las medidas que se utilizan en este documento para evaluar la calidad de los agrupamientos obtenidos.

[13] Una descripción detallada de este etiquetador se puede encontrar en http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/.

Medidas internas de calidad

Las medidas internas permiten evaluar la calidad del agrupamiento cuando no se tiene información de cuál es el grupo al que realmente pertenece cada documento. Estas medidas calculan un valor de calidad en función de la cohesión interna de los grupos, la cual tiene en cuenta qué tanto se parecen los documentos en un mismo grupo y qué tan diferentes son éstos de los documentos de otro grupo. En este documento se utiliza como medida interna la *similitud global* [73].

Para obtener esta medida es necesario calcular primero la similitud interna de cada grupo. Sea $C = \{C_1, C_2, \ldots, C_k\}$ un conjunto de grupos de documentos, la similitud interna de cada grupo C_i se calcula utilizando la expresión siguiente:

$$Similitud_{int}(C_i) = \frac{1}{|C_i|^2} \cdot \sum_{d,d' \in C_i} F(d, d'),$$

donde $|C_i|$ es la cantidad de documentos presentes en el grupo C_i y $F(d, d')$ es una función que determina la semejanza entre los documentos d y d'. Una de las funciones que se puede utilizar para el cálculo de la semejanza entre documentos es la medida del coseno [7].

Teniendo en cuenta la definición anterior, la similitud global del agrupamiento se calcula utilizando la siguiente expresión:

$$similitud_{global}(C) = \frac{\sum_{i=1}^{k} Similitud_{int}(C_i)}{k} \tag{9}$$

El valor de esta medida está en el intervalo [0,1] y entre mayor sea el valor de la misma mejor será la calidad del agrupamiento obtenido.

Es importante resaltar que, en nuestros experimentos, no se tendrá en cuenta para el cálculo de la similitud global según (9) a aquellos grupos formados por un único elemento. Los grupos formados por un sólo elemento tienen cohesión interna igual a 1 y por consecuente, un agrupamiento en el cual se formara un grupo unitario por cada documento de la colección obtendría el valor máximo para esta medida. Esta forma de proceder evita favorecer a algoritmos de agrupamiento que producen muchos grupos unitarios.

Otros ejemplos de medidas internas utilizadas son el *índice de Davies-Boulding* y el *índice de Dunn* [72] y la medida Λ [31].

Medidas externas de calidad

Este tipo de medida determina la calidad del agrupamiento comparando los grupos obtenidos con un conjunto de clases previamente definidas y que de manera general son determinadas manualmente por expertos. Ejemplos de estas medidas son *Fmeasure* [5],[51] y *Jaccard-index* [38].

La medida *Fmeasure* es una medida que combina las clásicas medidas de *precision* y *recall* para dar un indicador global de calidad del agrupamiento. Las expresiones para calcular esta medida se muestran a continuación:

$$Fmeasure = 2 \cdot \frac{precision \cdot recall}{precision + recall}, \tag{10}$$

donde *precision* y *recall* se definen como:

$$precision = \frac{n_{11}}{n_{11} + n_{10}}, \; recall = \frac{n_{11}}{n_{11} + n_{01}},$$

donde n_{10} representa la cantidad de pares de documentos que están en el mismo grupo y en diferentes clases, mientras que n_{01} es la cantidad de pares que están en diferentes grupos y en la misma clase. Esta medida permite, para cada par de documentos que comparten al menos un grupo en el agrupamiento, estimar cuándo esta predicción de asignarlos a un mismo grupo es correcta respecto a las clases predefinidas.

Es importante resaltar que esta definición de la medida *Fmeasure* fue propuesta en [5] específicamente para evaluar algoritmos que producen cubrimientos y la misma es una variante de la clásica medida $F1$ propuesta por van Rijsbergen en [77] que ha sido utilizada con frecuencia para evaluar la calidad de algoritmos que producen particiones.

El Jaccard index se denota por J y es una medida que establece una forma simple de comparar el agrupamiento resultante con el conjunto de clases predefinidas a través de la co-ocurrencia de los documentos en estos conjuntos. Esta medida se calcula utilizando la siguiente expresión:

$$J(A, B) = \frac{n_{11}}{\frac{N \cdot (N-1)}{2} - n_{00}}, \tag{11}$$

donde N es la cantidad de documentos de la colección, A representa el conjunto de grupos obtenidos, B el conjunto de clases definidas, n_{11} denota el número de pares de documentos que coinciden en un mismo grupo en A y en una misma clase en B, mientras n_{00} denota la cantidad de pares que se encuentran en diferentes grupos en A y diferentes clases en B. Los valores de esta medida están en el intervalo $[0, 1]$ y mientras mayor sea el valor de la misma, mejor es la calidad del agrupamiento.

En el caso de la medida Jaccard index, aunque ha sido utilizada para evaluar particiones, la forma en que se define sigue el mismo principio de la medida *Fmeasure* al utilizar para estimar la calidad del agrupamiento la cantidad de pares que comparten al menos un grupo en el conjunto de clases predefinidas y que en el agrupamiento resultante también comparten al menos un grupo. Esta forma de calcular la eficacia hace al Jaccard index independiente de la forma de los grupos y por lo tanto aplicable al problema del cubrimiento.

3.4. Experimentos

Para comparar la calidad de los grupos obtenidos por los diferentes algoritmos se utilizaron las medidas definidas en 3.3 y otros indicadores como el número de grupos generados y el promedio de elementos por grupo. Para cada algoritmo evaluado se ajustaron los parámetros de forma tal que se alcanzaran los mejores resultados.

En los experimentos se referirá a los algoritmos Extended Star, Generalized Star, Fuertemente Compacto Incremental, Single-link y Average-link por los nombres EStar, GStar, FCI, SLINK y ALINK respectivamente.

Experimento 1

En este experimento se comparó la calidad de los grupos obtenidos teniendo en cuenta las medidas externas. La comparación de los algoritmos CStar y CStar+ respecto a los algoritmos Star, EStar, GStar se muestra en las tablas 4 y 5; estas tablas muestran la calidad de cada algoritmo para diferentes valores del β.

La comparación entre los algoritmos CStar y CStar+ respecto a los algoritmos FTC, SLINK y ALINK se muestra en las figuras 10 y 11; en estas figuras el resultado mostrado corresponde al valor promedio de calidad obtenido.

Tabla 4. Evaluación considerando el Jaccard-index

		Valores de β					
Colección	**Algoritmo**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**	**Promedio**
AFP	Star	**0.44**	0.51	0.44	0.33	0.20	0.38
	EStar	0.34	0.52	0.50	0.40	**0.26**	0.40
	GStar	0.35	0.53	0.49	0.40	**0.26**	0.41
	CStar	0.36	**0.54**	**0.51**	**0.41**	**0.26**	**0.42**
	CStar+	0.36	**0.54**	**0.51**	**0.41**	**0.26**	**0.42**
Reuters	Star	0.42	0.33	0.31	0.26	0.21	0.31
	EStar	0.45	0.35	0.36	0.30	0.24	0.34
	GStar	0.45	0.36	**0.38**	0.31	**0.26**	0.35
	CStar	**0.46**	**0.41**	**0.38**	**0.32**	**0.26**	**0.37**
	CStar+	**0.46**	**0.41**	**0.38**	**0.32**	**0.26**	**0.37**

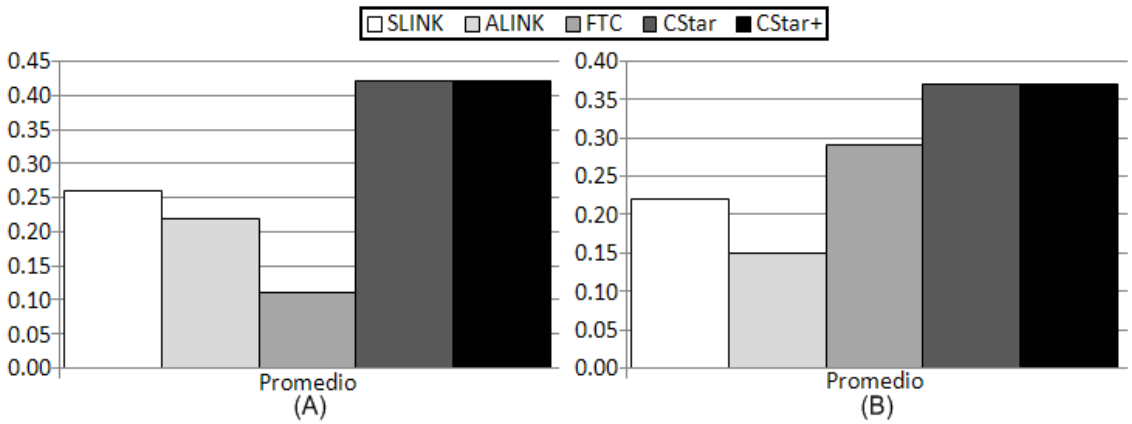

Figura 10. Evaluación considerando el Jaccard-index: (A) AFP, (B) *Reuters*

Tabla 5. Evaluación considerando el *Fmeasure*

		Valores de β					
Colección	**Algoritmo**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**	**Promedio**
AFP	Star	**0.61**	0.66	0.61	0.48	0.33	0.54
	EStar	0.51	0.67	0.64	**0.58**	**0.41**	0.56
	GStar	0.54	0.69	0.65	**0.58**	0.40	0.57
	CStar	0.54	**0.70**	**0.67**	**0.58**	**0.41**	**0.58**
	CStar+	0.54	**0.70**	**0.67**	**0.58**	**0.41**	**0.58**
Reuters	Star	0.60	0.50	0.48	0.40	0.35	0.47
	EStar	**0.63**	0.52	0.53	0.46	0.41	0.51
	GStar	**0.63**	0.51	0.54	0.47	**0.42**	0.51
	CStar	**0.63**	**0.59**	**0.55**	**0.49**	**0.42**	**0.54**
	CStar+	**0.63**	**0.59**	**0.55**	**0.49**	**0.42**	**0.54**

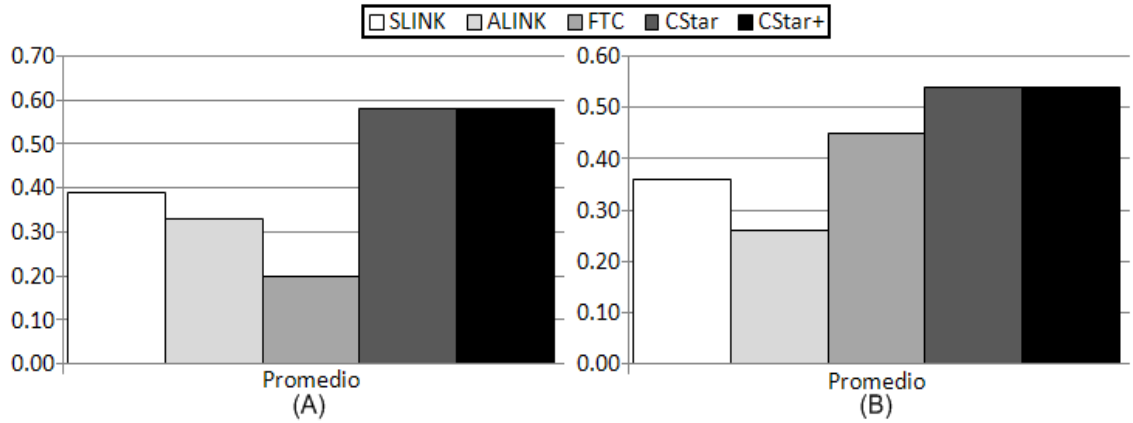

Figura 11. Evaluación considerando el *Fmeasure*: (*A*) AFP, (*B*) *Reuters*

Como se puede apreciar en las tablas 4 y 5, a excepción del caso en que el umbral de semejanza es 0.25 para la colección AFP, los algoritmos CStar y CStar+ obtienen en la mayoría de los casos mejores valores de calidad que los algoritmos Star, EStar y GStar. Si se considera la calidad promedio obtenida por cada algoritmo respecto al *Jaccard-index*, CStar y CStar+ representan un 10.53 % de mejora respecto al Star, un 5.0 % de mejora respecto al EStar y un 2.44 % de mejora respecto al GStar en la colección AFP; en la colección *Reuters*, los algoritmos CStar y CStar+ logran una mejora del 19.35 % respecto al Star, 8.82 % respecto al EStar y un 5.71 % respecto al GStar.

Si se tiene en cuenta los resultados para la medida *Fmeasure* en AFP, CStar y CStar+ obtienen un incremento en la calidad del 7.41 %, 3.57 % y 1.75 % respecto a los algoritmos Star, EStar y GStar respectivamente; en la colección *Reuters*, los incrementos son del 14.89 % respecto al algoritmo Star y 5.88 % respecto a los algoritmos EStar y GStar.

Aunque las mejoras obtenidas por CStar y CStar+ respecto a los algoritmos EStar y GStar en el caso promedio no son elevadas, sobre todo en la colección AFP, en el experimento 2 se muestra que los grupos obtenidos por CStar y CStar+ son más cohesionados que los obtenidos por EStar y GStar.

Por último, como se aprecia en las figuras 10 y 11, los valores de calidad obtenidos por CStar y CStar+ respecto a los algoritmos clásicos SLINK y ALINK son mucho mejores, sobre todo en la colección AFP. En el caso del algoritmo FTC, aunque alcanza un valor de calidad elevado respecto a ALINK y SLINK en la colección *Reuters*, los algoritmos CStar y CStar+ en esa misma colección lo superan en un 27.59 % respecto al *Jaccard-index* y en un 20.0 % considerando la medida *Fmeasure*.

Experimento 2

En este experimento se utiliza la medida interna para comparar la cohesión interna de los grupos formados por los algoritmos CStar y CStar+ respecto a la de aquellos formados por los algoritmos EStar y GStar. Los resultados de esta comparación en las colecciones AFP y *Reuters* se muestran en la tabla 6.

Tabla 6. Evaluación considerando la cohesión interna de los grupos

		Valores de β				
Colección	**Algoritmo**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**
AFP	EStar	0.273	0.347	0.401	0.458	0.500
	GStar	0.275	0.340	0.406	0.461	0.504
	CStar	**0.353**	**0.437**	**0.505**	**0.570**	**0.601**
	CStar+	**0.353**	**0.437**	**0.505**	**0.570**	**0.601**
Reuters	EStar	0.256	0.319	0.391	0.455	0.518
	GStar	0.260	0.318	0.391	0.459	0.521
	CStar	**0.284**	**0.373**	**0.460**	**0.543**	**0.608**
	CStar+	**0.284**	**0.373**	**0.460**	**0.543**	**0.608**

Como se puede apreciar, CStar y CStar+ obtienen en todos los casos grupos con una cohesión interna superior a los que obtienen los algoritmos GStar y EStar. Es importante mencionar que los resultados obtenidos por cada algoritmo superan los valores de cohesión interna que presentan las clases calculadas manualmente por los expertos (AFP tiene 0.24 y *Reuters* tiene 0.23).

El resultado de los algoritmos CStar y CStar+, de acuerdo a esta medida, refuerza los resultados obtenidos en el experimento 1.

Experimento 3

En este experimento se comparan la cantidad de grupos obtenidos y el promedio de elementos de éstos teniendo en cuenta los grupos obtenidos por los algoritmos Star, EStar, GStar, FCI, CStar y CStar+ en ambas colecciones. Los resultados de esta comparación se muestran en las tablas 7 y 8.

Como se aprecia en la tabla 7, sólo el algoritmo GStar obtiene menos grupos que los algoritmos CStar y CStar+. Sin embargo, es importante mencionar que la diferencia en

Tabla 7. Evaluación considerando la cantidad de grupos generados

		Valores de β				
Colección	**Algoritmo**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**
AFP	Star	143	196	265	326	411
	EStar	114	179	258	317	397
	FCI	351	373	404	439	489
	GStar	**108**	**173**	**245**	**308**	**389**
	CStar	109	175	250	310	391
	CStar+	109	175	250	310	391
Reuters	Star	1060	1646	2351	2997	3650
	EStar	662	1203	1923	2622	3378
	FCI	4528	4695	4950	5252	5600
	GStar	**648**	**1153**	**1845**	**2522**	**3279**
	CStar	651	1155	1856	2544	3295
	CStar+	651	1155	1856	2544	3295

Tabla 8. Evaluación considerando el promedio de elementos por grupos

		Valores de β				
Colección	**Algoritmo**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**
AFP	Star	9.32	5.43	3.57	2.83	2.13
	EStar	18.61	11.43	6.85	4.92	3.42
	FCI	3.29	2.96	2.56	2.27	1.92
	GStar	18.73	11.54	7.10	5.01	3.43
	CStar	**18.92**	**12.10**	**7.23**	**5.03**	**3.55**
	CStar+	**18.92**	**12.10**	**7.23**	**5.03**	**3.55**
Reuters	Star	47.21	24.97	15.77	10.93	7.74
	EStar	176.27	72.74	40.40	23.34	16.61
	FCI	4.62	4.39	4.04	3.68	3.34
	GStar	**225.36**	**87.29**	**49.28**	24.07	19.32
	CStar	218.03	83.73	46.98	**27.17**	**20.64**
	CStar+	218.03	83.73	46.98	**27.17**	**20.64**

cuanto al número de grupos es, en el peor caso, sólo de 22 grupos para el valor de β=0.45 en la colección *Reuters*.

Si se considera el promedio de elementos por grupos, como se puede apreciar en la tabla 8, los algoritmos CStar y CStar+ obtienen grupos con mayor cantidad de elementos, solamente superados por el algoritmo GStar en la colección *Reuters* para los valores de β iguales a 0.25, 0.30 y 0.35.

Es importante mencionar que, respecto al promedio de elementos por grupo que poseen las clases etiquetadas manualmente de las colecciones AFP (28,44) y *Reuters* (110,261), los mejores resultados los obtienen los algoritmos CStar y CStar+ para el valor de β=0.25 en el caso de AFP y el algoritmo GStar en el caso de *Reuters*.

Experimento 4

Para mostrar que CStar, a pesar de ser dependiente del orden de análisis de los elementos, reduce la influencia de esta dependencia en la calidad del agrupamiento resultante, se realizaron 10 ejecuciones de CStar para cada valor de β variando el orden de los documentos en las colecciones. Posteriormente, se calculó la calidad de cada agrupamiento resultante utilizando la medida interna y las medidas externas utilizadas en los experimentos 1 y 2. Utilizando estos valores de calidad se determinó la desviación estándard presente en los resultados para cada valor de β. El resultado de este experimento se puede observar en la tabla 9.

Tabla 9. Desviación estándar presente en los resultados

		Valores de β				
Colección	**Medidas**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**
AFP	Jaccard-index	1.05E-05	1.04E-03	1.14E-03	7.25E-05	9.01E-04
	Fmeasure	1.58E-05	1.08E-03	1.17E-16	7.91E-05	4.31E-04
	Sim. Global	4.83E-04	6.67E-04	1.40E-03	6.32E-04	5.16E-04
Reuters	Jaccard-index	1.58E-03	2.00E-03	8.41E-04	6.05E-04	1.24E-03
	Fmeasure	1.34E-03	1.33E-03	5.89E-04	9.30E-04	6.54E-04
	Sim. Global	5.16E-04	7.38E-04	4.83E-04	5.68E-04	4.22E-04

Como se puede apreciar en la tabla 9, la desviación estándar de los resultados es muy pequeña para cada valor de β, evidenciándose que aunque los agrupamientos sean diferentes, la calidad de éstos es muy similar.

4. Conclusiones

Los algoritmos de agrupamiento constituyen una herramienta útil en diferentes contextos. En el caso específico del procesamiento de documentos se han desarrollado diversos algoritmos bajo diferentes criterios. De forma general, como se pudo apreciar en la sección 2, estos algoritmos presentan algunos problemas .

Los problemas o limitaciones se pueden resumir en los siguientes aspectos:

- Los grupos formados presentan encadenamientos. Esta limitación afecta la cohesión interna de los grupos, ya que a un mismo grupo pueden ser asignados documentos que tengan muy baja semejanza, pero que se encuentren asociados transitivamente a través de otros documentos muy semejantes. Ejemplos de algoritmos que presentan este problema son: Single-link, STC, GLC, Compacto Incremental y ICT.
- Necesitan determinar a *priori* el número de grupos a obtener. En la mayoría de los casos, se desconoce la cantidad de clases en las que se distribuyen los documentos de la colección; luego, fijar un valor para la cantidad de grupos puede disminuir la cohesión interna de los grupos al asignar documentos poco semejantes a un mismo grupo. Ejemplos de algoritmos que presentan esta limitación son: K-means, NMF y Scatter/Gather.
- Requieren determinar a *priori* valores para varios parámetros[14] (diferentes del número de grupos). De forma similar al punto anterior, estos parámetros son desconocidos a *priori* para cada colección de documentos que se desee agrupar y la adecuada selección de los mismos determina la calidad del agrupamiento obtenido. Ejemplos de algoritmos con esta limitación son: STC, DC-tree, FTC, FIHC, Scatter/Gather, ICT, UMASS y MVGD.
- Los grupos determinados dependen de la selección inicial de objetos. Ejemplos de algoritmos con esta limitación son: K-means, Bisecting K-means, Scatter/Gather y los algoritmos genéticos.
- Obtienen muchos grupos, en algunos casos con pocos elementos, respecto a otros algoritmos para los mismos valores de los parámetros de entrada. Esta limitación dificulta el análisis de los resultados. Ejemplos de algoritmos con este problema son: Compacto Incremental, Fuertemente Compacto Incremental, Star y Extended Star.
- Pueden dejar documentos sin agrupar; es decir, elementos que no pertenecen a ningún grupo de la solución. Ejemplos de algoritmos con este problema son: DC-tree, FIHC y Extended Star.

Además de las limitaciones anteriores, existen algoritmos cuyo proceso de agrupamiento resulta muy costoso y que por lo tanto, no pueden aplicarse o resultarían ineficientes con colecciones muy grandes o de gran dimensionalidad, por lo que sólo se han aplicado a

[14] En este punto se asumió que el mínimo de parámetros que puede necesitar un algoritmo de agrupamiento para trabajar es 1

resúmenes, *snippets* o simplemente a documentos en los cuales se realiza un proceso de selección de términos. Ejemplos de algoritmos con este problema son: MVGD, DC-tree y STC.

Otro problema, presente en la mayoría de los algoritmos descritos, es que pueden obtener diferentes agrupamientos cuando se ejecutan sobre una misma colección, producto de la dependencia del orden de análisis de los documentos, el proceso de asignación de elementos a los grupos o simplemente del criterio de agrupamiento utilizado; por ejemplo: Star, CStar, CStar+, Single-Pass, etc. Existen algoritmos como GLC, Compacto Incremental, Fuertemente Compacto Incremental y Extended que solucionan esta dependencia; sin embargo, presentan otros problemas como encadenamiento, dejan elementos sin agrupar, etc.

No obstante a lo anterior, existen trabajos que plantean que la dependencia del orden de análisis de los documentos es una característica de interés más que una deficiencia, al permitir explorar las diferentes formas en que se pueden agrupar los elementos sin aumentar la cardinalidad del agrupamiento [4]. Otros autores plantean que el problema real acerca de la dependencia es si ésta afecta o no la calidad del agrupamiento; es decir, el problema no es que se obtengan diferentes agrupamientos sino cuánto varía la calidad de éstos [63]. Respecto a esto, la experimentación realizada demostró que los algoritmos CStar y CStar+, además de reducir limitaciones de algoritmos anteriores, obtienen agrupamientos que pueden ser diferentes pero cuyas respectivas calidades son muy cercanas.

Como se mencionó en la sección 1, es común que un documento aborde varios temas o tópicos, lo cual debería permitir que se asigne dicho documento a varios grupos; sin embargo, la mayoría de los algoritmos sólo permiten la asignación de un documento a un único grupo. Algunos de los algoritmos que permiten obtener grupos solapados o con traslape son: Star, Extended Star, Fuertemente Compacto Incremental, Generalized Star y STC, los cuales presentan, como se explicó en la sección 2, varias limitaciones de acuerdo a diferentes aspectos. Aunque los algoritmos CStar y CStar+ reducen muchas de las limitaciones encontradas en los algoritmos anteriormente comentados, es necesario continuar desarrollando algoritmos que generen cubrimientos, reduzcan las limitaciones señaladas y obtengan elevados valores de calidad.

Referencias

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *In Proceedings of the 20*th *VLDB Conference*, pages 487–499, 1994.
2. M. Alexandrov, A. Gelbukh, and P. Rosso. An approach to clustering abstracts. *In Proceeding of NLDB 2005 Conference, LNCS 3513, Springer, Verlag*, page 275–285, 2005.
3. J. Aslam, K. Pelekhov K., and D. Rus. Static and dynamic information organization with star clusters. *In Proceedings of the seventh international conference on Information and knowledge management*, pages 208–217, 1998.
4. J. Aslam, E. Pelekhov, and D. Rus. The star clustering algorithm for static and dynamic information organization. *Journal of Graph Algorithms and Applications, Vol. 8, No. 1*, pages 95–129, 2004.
5. A. Banerjee, C. Krumpelman, S. Basu, R. Mooney, and J. Ghosh. Model based overlapping clustering. *In Proceedings of International Conference on Knowledge Discovery and Data Mining(KDD)*, pages 532–537, 2005.
6. F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In *Proceedings of the 8th Int. Conf. on Knowledge Discovery and Data Mining (KDD)'2002.*, pages 436–442, 2002.
7. M. Berry. *Survey of Text Mining, Clustering, Classification and Retrieval.* Springer-Verlag, 2004.
8. G. J. Bloy. Blind camera fingerprinting and image clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(3):532–534, March 2008.
9. T. Brants, F. Chen, and A. Farahat. A system for new event detection. *Proceedings of the 26th annual international ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR2003*, pages 330–337, 2003.
10. T. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics, 3*, pages 1–27, 1974.
11. J. Carbonell, Y. Yang, J. Lafferty, R.D. Brown, T. Pierce, and X. Liu. Cmu report on tdt-2: Segmentation, detection and tracking. In *Proceedings of DARPA Broadcast News Workshop*, pages 117–120, 1999.
12. A. Casillas, M. T. González de Lena, and R. Martínez. Document clustering into an unknown number of clusters using a genetic algorithm. *In Proceedings of the 6th International Conference of Text, Speech and Dialogue, TSD 2003*, pages 43–49, 2003.
13. M. Connell, A. Feng, G. Kumaran, H. Raghavan, C. Shah, and J. Allan. Umass at tdt 2004. *In TDT2004 Workshop*, 2004.
14. G. Cselle, K. Albrecht, and R. Wattenhofer. Buzztrack: Topic detection and tracking in email. *IUI2007*, pages 446–453, 2007.
15. D. Cutting, D. Karger, J. Pedersen, and J. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. *In 15th ACM SIGIR*, pages 318–329, 1992.
16. D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Trans. Patt. Anal. Mach. Intell. 1*, pages 224–227, 1979.
17. B. C. M. Fung and et al. Hierarchical document clustering using frequent itemsets. In *Proceedings of the SIAM International Conference on Data Mining*, pages 59–70, 2003.
18. A. Gago-Alonso, A. Pérez-Suárez, and J. Medina-Pagola. Acons: a new algorithm for clustering documents. *In Proc. of XII CIARP, LNCS 4756, Springer-Verlag Berlin Heidelberg, CIARP2007*, pages 664–673, 2007.
19. G. García-Delgado. Un estudio sobre los algoritmos para el agrupamiento de documentos y herramienta de evaluación. Tesis de Licenciatura. Universidad de la Habana. Cuba, 2008.
20. R. J. Gil-García. *Algoritmos de agrupamiento sobre grafos y su paralelización*. PhD thesis, Escuela Superior de Tecnología y Ciencias Experimentales, Departamento de Ingeniería y Ciencia de los Computadores, Universidad Jaume I, 2005.
21. R. J. Gil-García, J. M. Badía-Contelles, and A. Pons-Porrata. Extended star clustering algorithm. *In Proceedings of CIARP'03, LNCS 2905*, pages 480–487, 2003.

22. R. J. Gil-García, J. M. Badía-Contelles, and A. Pons-Porrata. Parallel algorithm for extended star clustering. *In Proceedings of CIARP'04, LNCS 3287*, pages 402–409, 2004.
23. E. Greengrass. Information retrieval: A survey. Ed Greengrass. DOD Technical Report TR-R52-008-001, 2001.
24. G. Gupta, A. Liu, and J. Ghosh. Automated hierarchical density shaving: A robust, automated clustering and visualization framework for large biological datasets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, March 2008.
25. D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology, chapter 6.* Cambridge University Press, 1997.
26. J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, 2000.
27. J. Han, T. Kim, and J. Choi. Web document clustering by using automatic keyphrase extraction. *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*, pages 56–59, 2007.
28. J. Hartigan. *Clustering Algorithms.* John Wiley & Sons, New York, NY, 1975.
29. J. Hershberger, N. Shrivastava, and S. Suri. Cluster hull: A technique for summarizing spatial data streams. *22nd International Conference on Data Engineering (ICDE'06)*, pages 138–140, 2006.
30. J. H. Holland. Adaptation in natural and artificial systems. Technical report, Ann Arbor, MI: The University of Michigan Press, 1975.
31. D. Ingaramo, D. Pinto, P. Rosso, and M. Errecalde. Evaluation of internal validity measures in short-text corpora. *In Proc. of the 9th Int. Conf. on Comput. Linguistics and Intelligent Text Processing, CICLing-2008, Springer-Verlag, LNCS(4919)*, pages 555–567, 2008.
32. A. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.
33. A. K. Jain, M.Ñ. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.
34. A. Kakes and O. Casas. *Teoría de grafos.* Editorial Pueblo y Educación, 1987.
35. G. Karypis. Cluto: A clustering toolkit. Technical report, University of Minnesota, Department of Computer Science, Minneapolis, MN 55455, 2002.
36. G. Karypis, E. Han, and V. Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. In *IEEE Computer, 32(8)*, page 68–75, 1999.
37. W. H. Kim and T. Sikora. Image denoising method using diffusion equation and edge map estimated with k-means clustering algorithm. *Eight International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS '07)*, pages 21–24, 2007.
38. L. Kuncheva and S. Hadjitodorov. Using diversity in cluster ensembles. *In Proceedings of IEEE SMC 2004, The Netherlands*, pages 1214– 1219, 2004.
39. I. O. Kyrgyzov, H. Maitre, and M. Campedel. A method of clustering combination applied to satellite image analysis. *14th International Conference on Image Analysis and Processing (ICIAP 2007)*, pages 81–86, 2007.
40. A.N. Langville and C. D. Meyer. Google's pagerank and beyond: The science of search engine rankings. *Princeton University Press. ISBN 0-691-12202-4*, 2006.
41. B. Larsen and C. Aone. Fast and effective text mining using linear-time document clustering. In *KDD'99*, pages 16–22, 1999.
42. E. Levner, D. Pinto, P. Rosso, D. Alcaide, and R. Sharma. Fuzzifying clustering algorithms: The case study of majorclust. *In Proceeding of the 6th Mexican International Conference on Artificial Intelligence, MICAI-2007, Springer-Verlag, LNAI (4827)*, pages 821–830, 2007.
43. D. Lewis. Reuters-21578 text categorization text collection 1.0. http://kdd.ics.uci.edu.
44. P. Mahata. Exploratory consensus of hierarchical clusterings for melanoma and breast cancer. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, March 2008.
45. Y. Man-Quan, Luo Wei-Hua, Z. Zhao-Tao, and B. Shuo. Ict's approaches to htd and tracking at tdt2004. *In TDT2004 Workshop*, 2004.

46. J. F. Martínez-Trinidad, J. Ruiz-Shulcloper, and M. Lazo-Cortés. Structuralization of universes. *Fuzzy Sets and Systems, vol. 112 (3)*, pages 485–500, 2000.
47. H. Matsuda, T. Ishihara, and A. Hashimoto. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theoretical Computer Science*, 210(2):305–325, 1999.
48. R.T. Ng and J. Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, pages 1003–1016, 2002.
49. B. On, E. Elmacioglu, D. Lee, J. Kang, and J. Pei. Improving grouped-entity resolution using quasi-cliques. *Sixth International Conference on Data Mining, ICDM2006*, pages 1008–1015, 2006.
50. G. Pallis, L. Angelis, and A. Vakali. Model-based cluster analysis for web users sessions. *Proc. 15th Int'l Symp. Methodologies for Intelligent Systems (ISMIS'05)*, pages 219–227, 2005.
51. A. Peñas, A. Rodrigo, V. Sama, and F. Verdejo. Overview of the answer validation exercise 2006. *In Working Notes for the CLEF 2006 Workshop. Alicante, Spain*, 2006.
52. J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. *In Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 228 – 238, 2005.
53. J. Peng and J. Zhu. Refining spherical k-means for clustering documents. In *International Joint Conference on Neural Networks (IJCNN '06)*, pages 4146–4150, 2006.
54. S. Petridou, V. Koutsonikola, A. Vakali, and G. Papadimitriou. A divergence-oriented approach for web users clustering. *Proc. Int'l Conf. Computational Science and Its Applications (ICCSA '06)*, pages 1229–1238, 2006.
55. S. Petridou, V. A. Koutsonikola, A. I. Vakali, and G. I. Papadimitriou. Time-aware web users'clustering. *IEEE Transactions on Knowledge and Data Engineering*, pages 653–667, 2008.
56. D. Pinto and P. Rosso. On the relative hardness of clustering corpora. *In Proceedings of the 10th Int. Conf. on Text, Speech and Dialogue, TSD-2007, Springer-Verlag, LNAI (4629)*, pages 155–161, 2007.
57. A. Pons-Porrata. *Algoritmos de agrupamiento para la detección de sucesos en noticias*. PhD thesis, Centro de Investigación en Computación, 2004.
58. A. Pons-Porrata, R. Berlanga-Llavori, and J. Ruiz-Shulcloper. On-line event and topic detection by using the compact sets clustering algorithm. *Journal of Intelligent and Fuzzy Systems, 3-4*, pages 185–194, 2002.
59. A. Pons-Porrata, R. Berlanga-Llavori, J. Ruiz-Shulcloper, and J. M. Pérez-Martínez. Jerartop: A new topic detection system. *In Proceedings of the 9th Iberoamerican Congress on Pattern Recognition,LNCS 3287, Springer Verlag*, pages 446–453, 2004.
60. A. Pons-Porrata, J. Ruiz-Shulcloper, R. Berlanga-Llavori, and Y. Santiesteban-Alganza. Un algoritmo incremental para la obtención de cubrimientos con datos mezclados. *Reconocimiento de Patrones. Avances y Perspectivas. Research on Computing Science, CIARP'2002*, pages 405–416, 2002.
61. A. Pérez-Suárez. Algoritmos de agrupamiento para la generación de cubrimientos en colecciones de documentos. Master's thesis, Coordinación de Ciencias Computacionales, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), 2008.
62. A. Pérez-Suárez, J. F. Martínez-Trinidad, J. A. Carrasco-Ochoa, and J. Medina-Pagola. A new graph--based algorithm for clustering documents. *Accepted to be published in Proceedings of Foundation of Data Mining Workshop, FDM-ICDM2008 Workshops.*
63. A. Pérez-Suárez and J.E Medina-Pagola. A clustering algorithm based on generalized stars. *In Proceedings of the 5th International Conference on Machine Learning and Data Mining in Pattern Recognition, MLDM 2007, LNAI 4571, Leipzig, Germany, July 18-20*, pages 248–262, 2007.
64. G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.
65. F. Shahnaz. Clustering method based on nonnegative matrix factorization for text mining. Master's thesis, Department of Computer Science, University of Tennessee, Knoxville, TN, 2004.
66. Z. Shi, , and M. Ester. Performance improvement for frequent term-based text clustering algorithm. In *Research Project of CMPT 843*, 2003.

67. R. Sibson. An optimally effcient algorithm for the single link cluster method. In *Computer Journal, 16*, pages 30–34, 1973.
68. G. Sánchez-Díaz. *Desarrollo de algoritmos para el agrupamiento de grandes volúmenes de datos mezclados*. PhD thesis, Centro de Investigación en Computación, 2001.
69. W. Song and S. C. Park. Genetic algorithm-based text clustering technique: Automatic evolution of clusters with high efficiency. *In Proceedings of the Seventh International Conference on Web-Age Information Management Workshops (WAIMW'06)*, page 17, 2006.
70. M. Spitters and W. Kraaij. Tno at tdt2001: Language model-based topic detection. In *Topic Detection and Tracking (TDT) Workshop*, November 2001.
71. B. Stein and M. Busch. Density-based cluster algorithms in low-dimensional and high-dimensional applications. *In Proceedings of Second International Workshop on Text-Based Information Retrieval, TIR05*, pages 45–56, 2005.
72. B. Stein, S. Meyer, and F. Witbrock. On cluster validity and the information need of users. *In Proceedings of the 3rd IASTED*, pages 216–221, 2003.
73. M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, 2000.
74. TDT. The 2004 topic detection and tracking (tdt2004). *Task Definition and Evaluation Plan, version 1.0*, 2004.
75. TREC. Text retrieval conference. http://trec.nist.gov.
76. Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
77. C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, second edition, 1979.
78. M. Vignes and F. Forbes. Gene clustering via integrated markov models combining individual and pairwise features. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, August 2007.
79. E.M. Voorhees. Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. In *Information Processing and Management, 22*, pages 465–476, 1986.
80. W. Wai-chiu and A. W. Fu. Incremental document clustering for web page classification. *In IEEE 2000 International Conference on Information Society in the 21st Century: Emerging technologies and new challenges*, 2000.
81. K. L. Wu, C. Aggarwal, and P. Yu. Personalization with dynamic profiler. *Proc. Third Int'l Workshop Advanced Issues of E-commerce and Web-Based Information Systems (WECWIS '01)*, pages 12–20, 2001.
82. W. Xu, X. Liu, and Y. Gong. Document-clustering based on non-negative matrix factorization. *In Proceedings of SIGIR'03*, pages 267–273, 2003.
83. O. R. ZaIane and C. H. Lee. Clustering spatial data when facing physical constraints. *Second IEEE International Conference on Data Mining (ICDM'02)*, pages 737–740, 2002.
84. O. Zamir and O. Etziony. Web document clustering: A feasibility demonstration. *In Proceedings of the 21st Annual International ACM SIGIR Conference*, pages 46–54, 1998.
85. Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *11th Conference of Information and Knowledge Management (CIKM)*, pages 515–524, 2002.
86. D. Zuckerman. Np-complete problems have a version that's hard to approximate. *In Proceedings of the Eight Annual Structure in Complexity Theory Conference, IEEE Computer Society*, page 305–312, 1993.

RT_005, Noviembre 2008
Aprobado por el Consejo Científico CENATAV

Editor: Lic. Margarita Ilisástigui Avilés
Diseño de Portada: DCG Matilde Galindo Sánchez
RNPS No. 2143
ISSN 2072-6260
Indicaciones para los Autores:
Seguir la plantilla que aparece en www.cenatav.co.cu
C E N A T A V
7ma. No. 21812 e/218 y 222, Rpto. Siboney, Playa;
Ciudad de La Habana. Cuba. C.P. 12200
Impreso en Cuba