RNPS No. 2143
ISSN 2072-6260
Versión Digital

SERIE GRIS

REPORTE TÉCNICO
Minería de Datos

Algoritmos para el agrupamiento conceptual de objetos

Airel Pérez Suárez y
José E. Medina Pagola

RT_024 **abril 2014**

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

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

RNPS No. 2143
ISSN 2072-6260
Versión Digital

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

REPORTE TÉCNICO
Minería de Datos

SERIE GRIS

Algoritmos para el agrupamiento conceptual de objetos

Airel Pérez Suárez y
José E. Medina Pagola

RT_024 **abril 2014**

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

Tabla de contenido

Algoritmos para el agrupamiento conceptual de objetos

Airel Pérez Suárez y José E. Medina Pagola

Equipo de Investigaciones de Minería de Datos, Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV),
La Habana, Cuba
{asuarez,jmedina}@cenatav.co.cu

RT_024, Serie Gris, CENATAV
Aceptado: 17 de Marzo de 2014

Resumen. El agrupamiento es una de las técnicas fundamentales de la Minería de Datos y el Reconocimiento de Patrones, que ha sido utilizada satisfactoriamente en una gran variedad de contextos. No obstante, la mayoría de los algoritmos de agrupamiento desarrollados se centran solo en obtener una estructuración de la colección de objetos y dejan la interpretación de los grupos como tarea del usuario. En este reporte se hace un análisis crítico de los principales trabajos publicados hasta la fecha en el área del agrupamiento conceptual de objetos. Adicionalmente, se presenta una comparación cualitativa de estos algoritmos respecto a un conjunto de características deseables en este tipo de algoritmos de agrupamiento y se identifican algunas líneas de investigación que son necesarias desarrollar en el contexto del agrupamiento conceptual de objetos.

Palabras clave: minería de datos, agrupamiento, agrupamiento conceptual, formación de conceptos.

Abstract. Clustering is one of the fundamental techniques in Data Mining and Pattern Recognition that has been successfully applied in several contexts. However, most of the clustering algorithm developed so far have been focused only in organizing the collection of objects in a set of clusters, leaving the interpretation of those clusters to the user. In this technical report we present a critical review of the most influential articles reported in the field of conceptual clustering. Additionally, we present qualitative comparison of these algorithms regarding a set of desirable characteristics in this kind of clustering algorithms and also, we identify some research lines that need to be developed in the context of conceptual clustering.

Keywords: data mining, clustering, conceptual clustering, concept formation.

1. Introducción

El agrupamiento es una de las técnicas más importantes dentro de la Minería de Datos y el Reconocimiento de Patrones. Esta técnica se encarga de organizar una colección de objetos en clases o grupos, de forma tal que los objetos pertenecientes a un mismo grupo sean lo suficientemente similares como para poder inferir que son del mismo tipo y los objetos pertenecientes a grupos distintos sean lo suficientemente diferentes como para poder afirmar que son de tipos diferentes [1]. Existen varias áreas en las cuales los algoritmos de agrupamiento han sido utilizados satisfactoriamente, por ejemplo: en la medicina [2,3], en la segmentación de imágenes [4,5], en la biología [6] y en el análisis de huellas digitales [7], entre otras.

De forma general, se asume que los objetos a procesar por un algoritmo de agrupamiento están descritos por m rasgos o variables, cuyos valores pueden ser cuantitativos y/o cualitativos. A su vez, las variables pueden ser univaluadas, si toman un solo valor del dominio de valores posibles, o multivaluadas, si pueden

tomar mas de un valor. Adicionalmente, existen problemas en los que se puede desconocer el valor para algunos de los rasgos (i.e., rasgos incompletos) para uno o más objetos[8].

Los algoritmos de agrupamiento pueden ser clasificados atendiendo a diferentes criterios [9]. Algunos de los criterios utilizados con mayor frecuencia son: a) el tipo de grupos que forma el algoritmo, b) la capacidad que tiene el algoritmo para procesar cambios en la colección, c) el tipo de estructuración que forma con los objetos y d) el enfoque utilizado para formar los grupos. Este trabajo está centrado en este último criterio. De acuerdo al mismo, los algoritmos de agrupamiento se clasifican en *clasificatorios* o *conceptuales*. Los algoritmos clasificatorios generalmente construyen los grupos a partir de las relaciones de semejanzas de los objetos y como resultado obtienen solo la estructuración de los objetos. Por otra parte, los algoritmos conceptuales son aquellos que, además de organizar la colección de objetos en grupos, se encargan de encontrar una descripción intencional de los grupos. De forma general, el estudio de los métodos de agrupamiento se ha centrado más en los métodos clasificatorios. Estos métodos tienen como inconveniente el hecho de que dejan al especialista la tarea de interpretar los agrupamientos. Esto es una limitación para muchas aplicaciones en las que, además de conocer los grupos, los usuarios requieren conocer el significado subyacente de los mismos. En el resto del documento se hablará indistintamente de algoritmo de agrupamiento conceptual y algoritmo conceptual para referirse al mismo término.

En la literatura existen reportados varios algoritmos de agrupamiento conceptual. No obstante, a pesar de los resultados alcanzados hasta el momento en el estudio de este tipo de algoritmos, las aproximaciones existentes presentan limitaciones que dificultan su uso en problemas reales o son incapaces de responder a nuevos requerimientos. Estas razones hacen posible que el desarrollo de algoritmos de agrupamiento conceptual continúe siendo objeto de interés hoy en día.

El presente trabajo tiene como objetivo describir los principales algoritmos de agrupamiento conceptual que se han publicado hasta el momento, realizando un análisis crítico de cada uno de ellos que permita identificar sus limitaciones. Adicionalmente, se realiza una comparación cualitativa de los algoritmos presentados en la sección 2, en términos de algunas características importantes que deberían estar presentes en un algoritmo conceptual.

El resto de este documento está estructurado como sigue: en la sección 2 se analizan los algoritmos de agrupamiento conceptual más significativos del estado-del-arte. En la sección 3 se presenta una comparación cualitativa de los algoritmos descritos en la sección 2. Por último, en la sección 4 se presentan las conclusiones de este trabajo y se identifican algunas líneas de investigación que son necesarias desarrollar en el contexto del agrupamiento conceptual de objetos.

2. Algoritmos de agrupamiento conceptual

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos descritos en términos de un conjunto de atributos o rasgos $R = \{r_1, r_2, \ldots, r_m\}$. El problema del agrupamiento conceptual consiste en organizar la colección O en un conjunto de grupos $G = \{g_1, g_2, \ldots, g_k\}$, tal que se cumplan las siguientes condiciones:

a) Cada grupo $g_i \in G$ consta de dos partes: una descripción extensional que contiene los objetos de O incluidos en el grupo g_i y una descripción intencional, que contiene el o los conceptos que describen a los objetos incluidos en la descripción extensional de g_i, en términos del conjunto de rasgos R y utilizando un lenguaje determinado.
b) Todos los objetos de O tienen que pertenecer al menos a un grupo en G.

Aunque desde antes de 1970 ya se había manejado el término de formación de conceptos [10], los autores en el área del agrupamiento conceptual reconocen como precursor de esta línea a Ryszard S. Michalski, a partir de sus trabajos en la década de 1980 con el algoritmo CLUSTER/PAF [11]. Desde esa

fecha y hasta la actualidad se han reportados varios algoritmos de agrupamiento conceptual que, de forma general, siguen una de las tres aproximaciones siguientes: i) construyen primero la descripción extensional de los grupos y con base en esta, la descripción intencional (conceptos) de los grupos, ii) construyen primero un conjunto de conceptos presentes en la colección y posteriormente, forman la descripción extensional de cada grupo considerando los objetos que más se ajustan a dichos conceptos o iii) construyen a la misma vez la descripción extensional e intencional de los grupos.

En esta sección se describen algunos de los trabajos más influyentes en el contexto del agrupamiento conceptual, mencionándose las limitaciones presentes en cada uno de ellos.

2.1. Familia de algoritmos CLUSTER

En esta subsección se describen los algoritmos CLUSTER/PAF [11,12], CLUSTER/2 [13], CLUSTER/S [14] y CLUSTER/3 [15]. Primeramente se introducen un conjunto de conceptos necesarios para comprender estos algoritmos y posteriormente, se describen cada uno de ellos.

2.1.1. Conceptos preliminares

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos, descritos en términos de un conjunto de atributos o rasgos discretos $R = \{r_1, r_2, \ldots, r_m\}$, donde el dominio de valores de cada rasgo $r_j \in R$ se asume como finito y se representa por D_j. Sea $U \supset O$ el conjunto de todos los posibles objetos que se pueden formar considerando el conjunto R y el dominio de valores de cada rasgo en $r_j \in R$.

Definición 1 (Distancia sintáctica entre un par de objetos). *La distancia sintáctica entre un par de objetos $o_i, o_j \in O$ se define como el número de rasgos de R en los cuales dichos objetos tienen diferentes valores.*

Definición 2 (Selector). *Sea $r_j \in R$ un rasgo, $S_j \subseteq D_j$ un subconjunto del dominio de valores asociado al rasgo r_j y #, un operador del conjunto $\{\neq, \geq, >, \leq, <, =\}$. Un selector es una expresión del tipo $[r_j \# S_j]$.*

Algunos ejemplos de selectores son [height=*tall*], [length$\geq$2] y [weight=2..5]. Se entiende que un objeto $o_i \in O$ cubre o satisface un selector $[r_j \# S_j]$, si el valor del rasgo r_j en el objeto o_i satisface la relación # con el conjunto S_j.

Definición 3 (l-complejo). *Se llamará l-complejo al producto lógico de dos o más selectores.*

Sea l un l-complejo. Se entenderá que un objeto $o_i \in O$ satisface a l, o equivalentemente que l cubre a o_i, si o_i satisface a cada uno de los selectores de l. Al conjunto de todos los objetos de O que satisfacen a un l-complejo se le conoce por *s-complejo*. Adicionalmente, la dispersión de un l-complejo l se define como el número de objetos en $U \setminus O$ que satisfacen a l.

Definición 4 (Grado de intersección entre un par de l-complejos). *El grado de intersección entre un par de l-complejos es el número total de selectores que queda en ambos l-complejos, luego de eliminar de estos a los pares de selectores cuyos dominios de valores asociados no se intersectan.*

Sea $l_1 = [x_2 = 2,3][x_4 = 3,5,7][x_5 = 2..5]$ y $l_2 = [x_1 = 3][x_2 = 1][x_4 = 5..12][x_5 = 1]$ dos l-complejos. Como se observa en estos l-complejos, los dominios de los pares se selectores asociados a las variables x_2 y x_5 no se intersectan. Por lo tanto, el grado de intersección entre l_1 y l_2 es 3.

Definición 5 (Refunion). *Sea $L = \{l_1, l_2, \ldots, l_k\}$ un conjunto de l-complejos (o de s-complejos). Se conocerá por Refunion de L, denotada por $RU(L)$, a la operación que transforma el conjunto L en el l-complejo l que tiene la dispersión mínima entre todos los l-complejos que cubren al mismo conjunto de objetos, que son cubiertos por el conjunto L.*

Definición 6 (Estrella de un objeto respecto a un conjunto). *Sea $o \in O$ y $F \subseteq O$. La estrella de o respecto a F se denota por $G(o,F)$ y es el conjunto de todos los l-complejos maximales, respecto a la inclusión, que cubren al objeto o pero no a objetos de F. Un l-complejo l es maximal respecto a la inclusión, dada una propiedad P, si no existe otro l-complejo l' con la propiedad P tal que $l \subset l'$.*

2.1.2. CLUSTER/PAF, CLUSTER/2, CLUSTER/S y CLUSTER/3

El algoritmo CLUSTER/PAF construye un conjunto de grupos disjuntos, que tienen como descripción intencional a los l-complejos que representan a los objetos de los grupos. Para obtener dicho agrupamiento CLUSTER/PAF emplea una estrategia compuesta de cinco etapas.

En la primera etapa se seleccionan aleatoriamente k objetos (semillas) de la colección O; k es un parámetro del algoritmo. Sea $E = \{e_1, e_2, \ldots, e_k\}$ el conjunto de objetos "semillas" seleccionados. En la segunda etapa, se construye la estrella de cada objeto $e_t \in E, t = 1 \ldots k$, respecto al conjunto $E \setminus \{e_t\}$. Por otro lado, la tercera etapa tiene como objetivo formar un conjunto de l-complejos $S = \{l_1, l_2, \ldots, l_k\}$ tal que $\forall t = 1 \ldots k, l_t \in G(e_t, E \setminus \{e_t\})$ o l_t es una modificación de algún l-complejo en $G(e_t, E \setminus \{e_t\})$ y se cumple que

a) S alcanza el mayor valor de una medida de calidad Q predeterminada, entre todos los conjuntos S que se pueden formar a partir de los conjuntos $G(e_t, E \setminus \{e_t\}), \forall t = 1 \ldots k$.
b) el conjunto $G = \{G_1, G_2, G_k\}$, donde $\forall t = 1 \ldots k$, G_t es el s-complejo asociado al l-complejo l_t, es una partición de la colección O.

Para formar el conjunto S se utiliza una estrategia compuesta de tres pasos. En el primer paso se forma un conjunto de l-complejos S', seleccionando de cada estrella $G(e_t, E \setminus \{e_t\}), t = 1 \ldots k$, un l-complejo l_t. Si el conjunto de s-complejos asociado a los l-complejos de S' cumple la condición b), entonces se evalúa la calidad del conjunto S' utilizando la medida Q. Si la calidad de S' mejora el mejor resultado alcanzado hasta el momento, entonces $S = S'$ y se repite este procedimiento desde el paso 1, seleccionando otro conjunto S'. En otro caso, si no cumple la condición b), se trata de modificar los l-complejos de S' utilizando un procedimiento llamado NID. Si el procedimiento NID pudo modificar los l-complejos de S', de forma que este cumple la condición b), entonces se evalúa la calidad del conjunto S' utilizando la medida Q. Si la calidad de S' mejora el mejor resultado alcanzado hasta el momento, entonces $S = S'$ y se repite este procedimiento desde el paso 1, seleccionando otro conjunto S'.

Posteriormente, en la cuarta etapa se compara la calidad del conjunto S, de acuerdo a la medida Q, respecto a la calidad del mejor conjunto encontrado hasta ese momento (S_{opt}). Si la calidad de S es mejor que la de S_{opt}, entonces $S_{opt} = S$. Por último, en la quinta etapa se selecciona un nuevo conjunto E con base en los s-complejos de O que cubren a cada l-complejo de S. Si la iteración actual es impar, entonces el conjunto E se forma seleccionando de cada s-complejo s_t asociado al l-complejo $l_t \in S$, un objeto centro en s_t, de acuerdo a una distancia sintáctica. En otro caso, si la iteración es par, el conjunto E se forma seleccionando de cada s-complejo s_t el objeto que esté más lejos del objeto centro de s_t. A continuación, se repite todo este proceso desde la etapa 2, hasta que se realicen un número de iteraciones predefinidas sin hallar un conjunto S que mejore a S_{opt}.

Para evaluar la calidad de un conjunto de l-complejos $S = \{l_1, l_2, \ldots, l_k\}$, el algoritmo CLUSTER/PAF utiliza una función llamada "Funcional de evaluación lexicográfico" (LEF, por sus siglas en inglés). La

función LEF mide la calidad del conjunto S atendiendo a: i) la suma de la dispersión de cada uno de los l-complejos de S, ii) el promedio del grado de intersección entre todos los pares de l-complejos de S, iii) la variabilidad del tamaño de los s-complejos asociados a cada l-complejo de S y iv) el número total de rasgos incluidos en los l-complejos de S. Mientras menor valor de cada uno de estos parámetros tenga un conjunto de l-complejos, mejor será este.

El procedimiento NID tiene como objetivo transformar los l-complejos de un conjunto S, de forma tal que el conjunto de s-complejos asociado a S sea disjunto. Este procedimiento está compuesto de seis pasos. En el primer paso se forma para cada l-complejo $l_t \in S'$, el conjunto $CORE_{l_t}$ que contiene a los objetos de O que solo satisfacen al l-complejo l_t. En este paso también se forma el conjunto $Resto$, que contiene a los objetos de O que satisfacen a más de un l-complejo a la vez. En el segundo paso se aplica la operación Refunion a cada conjunto $RU(CORE_{l_t})$ y si para algún par de conjuntos $CORE_{l_t}$ y $CORE_{l_q}$ se cumple que $RU(CORE_{l_t}) \cap RU(CORE_{l_t}) \neq \emptyset$, entonces el conjunto S' no se puede volver disjunto y termina el procedimiento NID. En otro caso, en el tercer paso se extrae un objeto $o \in Resto$ y para cada l-complejo $RU(CORE_{l_t})$, se construye el l-complejo $l_t^{'} = RU(o \cap RU(CORE_{l_t}))$. Posteriormente, en el cuarto paso se eliminan todos los pares de l-complejos $l_t^{'}, l_q^{'}$ tal que $l_t^{'} \cap l_q^{'} \neq \emptyset$. Si no queda ningún otro l-complejo, entonces el conjunto S' no se puede volver disjunto y termina el procedimiento NID. En otro caso, en el quinto paso se selecciona el l-complejo $l_t^{'}$ que minimice la diferencia entre la dispersión de $l_t^{'}$ y l_t, el número de objetos de $Resto$ que cubren a $l_t^{'}$ y la diferencia entre el número de selectores de $l_t^{'}$ y l_t. Por último, en el sexto paso se sustituye l_t por $l_t^{'}$ y si $Resto \neq \emptyset$, entonces se repite todo este procedimiento desde el tercer paso.

El algoritmo CLUSTER/2 modifica ligeramente la estrategia empleada por CLUSTER/PAF para construir un conjunto de grupos disjuntos y sus descripciones. La primera modificación que introduce CLUSTER/2 es en la segunda etapa de CLUSTER/PAF, durante la formación de las estrellas. En esta modificación, además de restringir el número de l-complejos a formar para cada estrella, de acuerdo a un parámetro, también se le aplica la operación Refunion a cada uno de los l-complejos de cada estrella $G(e_t, E \setminus \{e_t\})$. Los l-complejos que se obtienen de la operación anterior, para cada estrella $G(e_t, E \setminus \{e_t\})$, son posteriormente procesados, utilizando un operador de nombre GEN. Este operador proviene de estudios realizados en el campo del aprendizaje inductivo y su objetivo es simplificar y generalizar los l-complejos mencionados anteriormente. En este contexto, simplificar un l-complejo se refiere a reducir el numero de selectores que describen al mismo. Por otra parte, generalizar un l-complejo l se refiere a modificar de cierta forma los selectores del mismo, para obtener un l-complejo l', de forma tal que el s-complejo asociado l' es un superconjunto del s-complejo asociado a l. Para procesar un l-complejo el operador GEN utiliza un conjunto de reglas que dependen del tipo de rasgo de cada selector de dicho l-complejo. Las reglas de generalización utilizadas pueden consultarse en [16]. Otra modificación que incluye CLUSTER/2 consiste en adicionar otro criterio a la función LEF que se utiliza para evaluar cada conjunto de l-complejos en la etapa cuatro. En este caso, además de los criterios que define CLUSTER/PAF, CLUSTER/2 utiliza también el índice de discriminación, que no es más que el número de rasgos que tienen valores diferentes en todos los l-complejos. Por último, CLUSTER/2 incluye también dos mejoras al procedimiento NID con el objetivo de agilizar su funcionamiento. Estas mejoras son: 1) ordenar los conjuntos S que se pueden formar en la etapa tres, en orden descendente de su valor de dispersión total y 2) generar las estrellas de la etapa dos dinámicamente; es decir, según se vaya procesando cada objeto y no todas las estrellas a la vez.

A diferencia de los algoritmos CLUSTER/PAF y CLUSTER/2, que asumen que los objetos se representan como una secuencia de valores de rasgos, el algoritmo CLUSTER/S asume que los objetos están representados por descripciones estructuradas. Este tipo de descripciones están expresadas mediante el cálculo de predicados anotados (APC, por sus siglas en inglés) [17]. Adicionalmente, CLUSTER/S permite que se entre como parámetro al algoritmo información externa que puede usar como parte del

agrupamiento. Esta información externa puede estar dada en forma de: i) reglas de inferencia, a partir de las cuales se puede generar nuevos rasgos para los objetos o ii) una red de dependencia de objetivo (GDM, por sus siglas en inglés), que específica cuáles atributos son los más importantes para el agrupamiento que se desea realizar. Para formar un agrupamiento conceptual CLUSTER/S emplea una estrategia compuesta de dos etapas. En la primera etapa se aplica el algoritmo INDUCE/2 [18] sobre la colección de objetos, para generar una descripción de cada objeto en forma de una tupla de atributos o rasgos. Posteriormente, en la segunda etapa se aplica el algoritmo CLUSTER/2 sobre esta nueva representación de cada objeto, para encontrar el conjunto de grupos y sus descripciones.

El algoritmo CLUSTER/3 asume que los objetos están descritos por tuplas de pares atributo-valor y a diferencia de sus predecesores, CLUSTER/3 permite formar una estructura jerárquica de h niveles; donde h es un parámetro del algoritmo. Para crear la jerarquía CLUSTER/3 utiliza una estrategia divisiva que, partiendo de un grupo que contiene a todos los objetos de la colección, forma el primer nivel utilizando el algoritmo CLUSTER/PAF sobre el nodo raíz. Si $h > 1$, los grupos del nivel h son obtenidos aplicando el algoritmo CLUSTER/PAF sobre los objetos incluidos en cada grupo del nivel $h-1$. El número de grupos k a obtener, cada vez que se aplica el algoritmo CLUSTER/PAF, es un parámetro o es determinado por el algoritmo. Para determinar el valor de k, se aplica el algoritmo CLUSTER/PAF para los valores de $k = 2 \ldots MAX$; donde MAX es un parámetro del algoritmo. Cada uno de los agrupamientos obtenidos son evaluados utilizando la medida LEF y aquel que maximice el valor de esta medida es seleccionado. Adicionalmente, el algoritmo CLUSTER/3 permite incorporar información externa al proceso de agrupamiento, a través de la seleccionando del conjunto de atributos que, de acuerdo a información externa brindada por por el usuario, sean relevantes al problema a resolver.

La principal limitación de los algoritmos CLUSTER/PAF, CLUSTER/2, CLUSTER/S y CLUSTER/3 es su complejidad computacional, que puede ser exponencial debido principalmente al procedimiento NID. Esta alta complejidad hace a estos algoritmos poco útiles en problemas reales con colecciones de muchos objetos. Otra limitación de estos algoritmos es que requieren ajustar valores de varios parámetros (cuatro o más en cada algoritmo), los cuales dependen de la colección a procesar. El ajuste de estos parámetros generalmente es *ad-hoc* y puede resultar difícil en problemas reales. Por último, estos algoritmos dependen del orden de selección de las k semillas y no permiten procesar objetos con rasgos incompletos.

2.2. WITT

El algoritmo WITT [19] es un modelo computacional que pretende simular la forma en que los seres humanos categorizan un conjunto de objetos, a partir de las observaciones previas de éstos. Este algoritmo asume que los objetos están descritos por pares atributo-valor, siendo los atributos variables booleanas o k-valentes.

WITT basa su funcionamiento en las interrelaciones que existen entre los atributos que describen a los objetos y describe a cada grupo en términos de las co-ocurrencias de pares de valores de estos atributos, expresadas a través de tablas de contingencia. Una tabla de contingencia es una matriz que muestra, para un par de atributos, la frecuencia de co-ocurrencia de cada posible par de valores de dichos atributos.

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos, descritos en términos de un conjunto de atributos o rasgos discretos $R = \{r_1, r_2, \ldots, r_m\}$, donde el dominio de valores de cada rasgo $r_j \in R$ es finito y se representa por D_j.

Para la construcción de un agrupamiento disjunto, este algoritmo emplea dos etapas: inicialización y refinamiento. La etapa de inicialización sigue una estrategia similar a la que emplea el algoritmo Single-Link [20]. En el primer paso de esta etapa se determina el par de objetos más cercanos, de acuerdo a una medida de distancia entre los objetos, y se forma un grupo con dicho par de objetos. Sea d_{min} la menor

distancia existente entre los objetos de la colección. Posteriormente, en el segundo paso se determina el par de elementos, siendo un elemento un objeto o un grupo, que tiene la menor distancia. Si la distancia entre este par de elementos es menor que $f \cdot d_{min}$, donde f es un parámetro del algoritmo, entonces en el tercer paso se unen dichos elementos y se repite este procedimiento desde el segundo paso. En otro caso, si la menor distancia es mayor que $f \cdot d_{min}$, entonces se termina la primera etapa. Esta unión puede hacer que: a) se forme un nuevo grupo si los dos elementos eran objetos, b) se adicione un elemento a un grupo si los elementos eran un grupo y un objeto, o c) se unan dos grupos si ambos elementos eran grupos.

Sea $C = \{C_1, C_2, \ldots, C_k\}$ el conjunto de grupos construidos en la primera etapa y $L = O \setminus \left(\bigcup_{i=1}^{k} C_i\right)$, el conjunto de objetos que quedaron sin agrupar al terminar dicha etapa. En la segunda etapa se realiza un proceso para asignar los objetos del conjunto L a algún grupo de C. Este proceso consta de cinco pasos. En el primer paso se calcula, para cada objeto $o_i \in L$, la *cohesión* que tuviera cada grupo $C_j \in C$ si se le adicionara el objeto o_i. En el segundo paso se selecciona el par objeto-grupo que alcanza la mayor cohesión y si este valor de cohesión supera un umbral t previamente establecido, entonces se elimina dicho objeto de L y se inserta en dicho grupo. El proceso anterior se repite desde el primer paso, mientras se inserten objetos a los grupos. La cohesión de un grupo C_i se denota por Coh_{C_i} y se calcula como el cociente entre la *cohesión intra-grupo* e *inter-grupo* que tiene el grupo C_i.

La cohesión intra-grupo se denota por $Coh_{C_i}^{In}$ y se calcula como sigue:

$$Coh_{C_i}^{In} = \frac{\sum_{j=1}^{m-1} \sum_{p=j+1}^{m} \Theta_{jp}}{\binom{m}{2}},$$

donde m es el número de atributos que describen a los objetos y Θ_{jp} es la distribución de co-ocurrencia de los atributos j y p, la cual se calcula de la siguiente forma:

$$\Theta_{jp} = \frac{\sum_{t=1}^{|D_j|} \sum_{q=1}^{|D_p|} f_{tq} \cdot \log f_{tq}}{\left(\sum_{t=1}^{|D_j|} \sum_{q=1}^{|D_p|} f_{tq}\right) \cdot \log \left(\sum_{t=1}^{|D_j|} \sum_{q=1}^{|D_p|} f_{tq}\right)},$$

donde f_{tp} es la frecuencia de co-ocurrencia del valor t-ésimo del atributo j y del valor q-ésimo del atributo p.

La cohesión inter-grupo se denota por $Coh_{C_i}^{Out}$ y se calcula como sigue

$$Coh_{C_i}^{Out} = \frac{\sum_{c \neq i} B_{ic}}{|C| - 1},$$

donde B_{ic} mide la varianza en las co-ocurrencias de los pares atributo-valor, en la unión de los grupos C_i y C_c, a través de la siguiente fórmula:

$$B_{ic} = \frac{1}{Coh_{C_i}^{In} + Coh_{C_c}^{In} - 2 \cdot Coh_{C_i \cup C_c}^{In}}.$$

Una vez que se termina el proceso de los primeros dos pasos, en el tercer paso se ejecuta la etapa de inicialización sobre el conjunto de objetos que quedan en P. Sea $C' = \{C_1', C_2', \ldots, C_h'\}$ el conjunto de grupos que se obtiene producto del paso anterior y P el conjunto de objetos de O que aún quedan sin agrupar, luego del tercer paso. Posteriormente, en el cuarto paso se recorre cada grupo de C_j' y se calcula la cohesión intra-grupo de cada uno de los grupos que se obtiene uniendo a C_j' con un grupo del conjunto C. Si la cohesión intra-grupo de cada uno de estos grupos determinados es mayor a un umbral t_1, entonces

se une el grupo $C_j^{'}$ al conjunto C. Si como resultado del cuarto paso, se unió al menos un grupo nuevo al conjunto C, entonces se repite el proceso anterior desde el primer paso, considerando el conjunto P de objetos que actualmente quedan sin agrupar. En otro caso, si no se unió ningún grupo nuevo a C, entonces en el quinto paso se determina del conjunto C el par de grupos C_i y C_j tal que $Coh^{In}_{C_i \cup C_j}$ sea máximo. Si este valor máximo supera el umbral t_1, entonces se unen dichos grupos y se repite el proceso anterior desde el primer paso, considerando el conjunto P de objetos que actualmente quedan sin agrupar. En otro caso, si este valor no supera t_1, termina el algoritmo. La complejidad computacional de WITT es $O(n^3 \cdot m)$.

En [19] se reporta también una variante incremental de WITT en la cual se especifica, a través de un parámetro Z, el número de objetos a procesar en cada iteración. En esta variante, WITT agrupa los primeros Z objetos utilizando la misma estrategia comentada anteriormente. Posteriormente, cada vez que WITT lee otro bloque de Z objetos, inserta dichos objetos en el conjunto P de objetos que no han sido agrupados y los procesa utilizando la segunda etapa. Adicionalmente, en [21] se realiza un análisis del algoritmo WITT y como consecuencia del mismo, se presenta una variante de éste que sigue la misma estructura general, pero que utiliza otras fórmulas para calcular la cohesión intra-grupo e inter-grupo de los grupos construidos por WITT.

La principal limitación del algoritmo WITT es que puede dejar objetos sin agrupar. Otra limitación es que requiere ajustar valores para tres parámetros (f, t_2 y t_3) los cuales dependen de la colección a procesar y pueden resultar complejos de ajustar en problemas reales. Por último, WITT no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

2.3. EPAM

EPAM [10] es un algoritmo conceptual que permite formar una red discriminativa a partir de un conjunto de objetos, descritos por atributos cualitativos. EPAM es un algoritmo que es precursor de las ideas posteriormente descritas en UNIMEN [22] y COBWEB [23]. De hecho, la red discriminativa es una idea que posteriormente dio lugar a las jerarquías de conceptos.

Una red discriminativa es una estructura tipo árbol, en la cual: i) cada nodo no terminal tiene asociado un atributo y ii) los enlaces de un nodo no terminal N a sus hijos, se etiquetan con el valor que tiene el atributo asociado a N, para los objetos almacenados en el sub-árbol definido por cada nodo hijo. Como caso específico, uno de estos enlaces se etiqueta como "otros", definiendo a todos aquellos valores del atributo que no son considerados en el resto de los nodos hijos. Adicionalmente, los nodos terminales de esta estructura cuentan con un concepto, formado por una lista de pares atributo-valor, que se espera describa al conjunto de objetos almacenados en dicho nodo. Los objetos almacenados en estos nodos forman un agrupamiento disjunto.

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos descritos por atributos cualitativos. Para formar la red discriminativa, en lo siguiente referido solo como red, EPAM procesa los objetos uno a uno y realiza un proceso de búsqueda sobre la red formada hasta ese momento. Este proceso está guiado por los valores de los atributos asociados en cada nodo. Sea $o_i \in O$ el objeto a adicionar. Como parte del proceso de búsqueda, EPAM compara en cada nodo N, el valor que tiene o_i respecto al atributo asociado a N, con cada una de las etiquetas asociadas a los nodos hijos de N. Si alguna de estas etiquetas concuerda con el valor que tiene o_i para el mismo atributo, entonces el proceso de búsqueda continúa por el nodo hijo asociado a dicha etiqueta. En otro caso, el proceso continúa por el nodo asociado a la etiqueta "otros". Una vez que este proceso de búsqueda llega a un nodo terminal, se compara el concepto que describe a dicho nodo con los valores que tiene o_i para el mismo conjunto de atributos que describen al concepto. Si todos los valores coinciden, entonces o_i es almacenado en dicho nodo y además, se lleva a cabo un proceso de *familiarización*. Este proceso tiene como objetivo aumentar la especialización del concepto que describe

al nodo, a través de la incorporación de un nuevo par atributo-valor contenido en o_i. En otro caso, si el concepto y el objeto difieren respecto al valor de un atributo, entonces se ejecuta un proceso de *distinción*. Este otro proceso tiene como objetivo ampliar la red construida hasta ese momento, de forma que pueda aceptar al nuevo objeto o_i.

En el proceso de familiarización, se selecciona al azar un par atributo-valor del objeto o_i que no esté en el concepto que describe al nodo terminal. Posteriormente, este par se incorpora a la descripción del concepto. Por otra parte, en el proceso de distinción es necesario recorrer otra vez la red, desde la raíz, para encontrar el primer nodo no terminal N' a partir del cual, difiere el valor que tiene o_i para el atributo asociado a N', respecto al valor de las etiquetas de los enlaces que tiene N' con sus nodos hijos; es decir, se pretende encontrar la primera rama etiquetada como "otros" que tomó el proceso de búsqueda anterior. Si dicho nodo no terminal existe, entonces se crean dos nuevos nodos terminales T_1 y T_2. El nodo T_1 contiene el objeto o_i y tiene como concepto el par atributo-valor que tiene o_i para el atributo asociado al nodo no terminal encontrado. Por otra parte, el nodo T_2 se crea vacío y tiene como concepto el par atributo-valor que tenía el nodo terminal encontrado en la primera búsqueda y en el cual difería con o_i. Posterior a la creación de T_1 y T_2, ambos nodos son adicionados como hijos del nodo no terminal encontrado y los enlaces a ellos son etiquetados con los valores que tienen T_1 y T_2 para el atributo asociado a su nodo padre. En otro caso, si no se encuentra el primer nodo no terminal que difirió con o_i, respecto al valor de su atributo asociado, y se llega en esta segunda búsqueda al mismo nodo terminal de la primera búsqueda, entonces EPAM emplea una estrategia compuesta de tres pasos para incorporar el objeto a la red.

En el primer paso de esta estrategia, EPAM crea un nuevo nodo no terminal M y asocia a este nodo un atributo no seleccionado previamente en el camino desde la raíz al nodo terminal encontrado en la primera búsqueda; sea este nodo terminal N'. En el segundo paso, se crea un nodo T que contiene a o_i y cuyo concepto se forma con todos los pares atributo-valor que permiten llegar desde la raíz de la red hasta N' y con el par atributo-valor que tiene o_i para el atributo asociado al nodo M. Posteriormente, en el tercer paso EPAM elimina al nodo N' y en su lugar coloca al nodo M, poniendo como hijos de este último a los nodos T y N'. La arista del nodo M a T se etiqueta con el valor que tiene o_i para el atributo asociado a M; la arista de M a N' se etiqueta como "otros".

Entre las limitaciones de EPAM se encuentra que solo permite procesar objetos descritos por atributos cualitativos. Otra limitación es que el proceso de construcción de la red y el de construcción de los conceptos de los nodos, dependen del orden de análisis de los objetos y del orden de análisis de los atributos. Adicionalmente, los conceptos generados para cada nodo pueden no cubrir a todos los objetos almacenados en dicho nodo. Por último, el proceso de construcción y mantenimiento de la red puede resultar computacionalmente costoso en problemas reales, que procesen colecciones de muchos objetos.

2.4. UNIMEN y RESEARCHER

UNIMEN [24,22] y RESEARCHER [24] son algoritmos conceptuales propuestos por Lebowitz, que forman una jerarquía de conceptos en forma incremental. A diferencia de otros algoritmos conceptuales, UNIMEN y RESEARCHER construyen a la vez la estructuración extensional e intencional de la colección. UNIMEN está diseñado para procesar objetos descritos por pares atributo-valor, donde los atributos pueden ser cualitativos o cuantitativos. Por otra parte, RESEARCHER está diseñado para procesar objetos estructurados; es decir, objetos que en su descripción contienen los valores de los atributos y las relaciones existentes entre ellos. Dado que la estructura general de ambos algoritmos es la misma, a continuación solo se describe UNIMEN y en donde sea necesario, se mencionarán las diferencias respecto a RESEARCHER.

UNIMEN basa su funcionamiento en una estructura de datos llamada GBM (Generalization-Based Memory) [24]. Esta estructura de datos es jerárquica y en la misma, los niveles superiores representan conceptos más generales y los niveles inferiores almacenan conceptos más específicos. Esta estructura está compuesta por nodos, cada uno de los cuales contiene un conjunto de objetos, el concepto que describe a dicho conjunto, una lista de nodos hijos de dicho nodo y un vector numérico que representa la confianza que se tiene en cada atributo del concepto que describe al nodo. Los conceptos almacenados en cada nodo se representan en el mismo espacio en el que se representan los objetos de la colección. En el caso de UNIMEN como pares atributo-valor y en el caso de RESEARCHER, como objetos estructurados (grafos o árboles).

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos. Para formar una jerarquía de conceptos a partir de O, UNIMEN procesa los objetos incrementalmente (uno a uno), siguiendo una estrategia compuesta de dos etapas. Sea $o_i \in O$ el objeto que se está procesando. En la primera etapa se realiza una búsqueda en profundidad sobre la GBM para encontrar los nodos con los conceptos más específicos, cuyas distancias al objeto o_i sean menor que un umbral predefinido. El proceso de búsqueda parte del nodo raíz y está compuesto de tres pasos. En el primer paso se calcula la distancia entre el concepto c almacenado en el nodo actual y el objeto o_i. La distancia entre un objeto o_i y un concepto c se calcula como la sumatoria de las distancias que existen entre o_i y c, respecto a cada uno de los atributos que los describen. Para cada atributo se define una función de distancia con valor en el intervalo $[0,1]$, en la cual 0 significa cotejo perfecto y 1 cotejo nulo. Si la distancia entre o_i y c es mayor que un umbral p_1, entonces se detiene la búsqueda por ese camino y se devuelve una lista vacía. En otro caso, si la distancia entre o_i y c es menor que p_1, entonces en el segundo paso se sigue la búsqueda recursivamente, por cada uno de los nodos hijos del nodo actual.

Cuando las búsquedas realizadas por los nodos hijos terminan, si para algún nodo hijo se obtuvo como resultado una lista no vacía, entonces en el tercer paso se devuelve, como resultado de la búsqueda en el nodo actual, la lista resultante de la unión de todas las listas no vacías obtenidas a partir de las búsquedas en los nodos hijos. En otro caso, si todas las búsquedas en los nodos hijos retornaron listas vacías, significa que el nodo actual es el nodo más específico que mejor coteja con o_i y por lo tanto, se devuelve una lista que contiene al nodo actual. Como se deduce de la descripción anterior, la búsqueda de nodos que realiza UNIMEN puede dar como resultado una lista con más de un nodo y por lo tanto, el objeto o_i podría adicionarse a más de un nodo de la jerarquía. De esta forma UNIMEN y RESEARCHER permiten formar jerarquías con traslape.

Una vez determinada la lista L de nodos más específicos que mejor cotejan con o_i, en la segunda etapa se recorre esta lista y se inserta el objeto o_i en cada uno de los nodos de l, empleando una estrategia compuesta de tres pasos. Sea $n_j \in L$ el nodo que se está procesando. En el primer paso, se determina el conjunto de atributos R que forman parte de la descripción de o_i, pero no del concepto que describe a los objetos almacenados en n_j. Posteriormente, en el segundo paso se determina la distancia de o_i a cada uno de los objetos almacenados en n_j. Sea T el conjunto de objetos almacenados en n_j cuya distancia a o_i es igual o menor que un umbral p_2. Si $T = \emptyset$, entonces en el tercer paso o_i es almacenado en el nodo n_j con el resto de los objetos. En otro caso, en el tercer paso se crea para cada objeto $o' \in T$ un nodo en la jerarquía. El nodo creado para cada objeto $o' \in T$ tiene como descripción extensional al par de objetos o', o_i. Para formar la descripción intencional de dichos nodos, se selecciona el conjunto de atributos cuyos valores en o' y en o_i estén a una distancia menor que un umbral p_3, de acuerdo con la función de distancia definida para cada rasgo. Definido el conjunto de atributos que formará el concepto del nodo creado, se define el valor de cada atributo considerando el tipo de atributo, así como los valores de dicho atributo en los objetos o' y o_i. Para los atributos numéricos se toma el valor promedio del atributo en los objetos o' y o_i. Para los atributos cualitativos se selecciona cualquiera de los valores del atributo en los objetos o' y o_i. Para atributos jerárquicamente ordenados se selecciona el menor ancestro común.

Dado que los conceptos que describen a los nodos se construyen con base en los atributos que describen a dos objetos, a menudo estos conceptos son muy específicos. Para evitar estos problemas UNIMEN incluye, dentro del proceso de búsqueda efectuado en la primera etapa, la evaluación del nodo que se está procesando, en términos del concepto contenido en el mismo. Esta evaluación consiste en: i) aumentar los valores de confianza de aquellos atributos que están en el concepto almacenado en el nodo y también en el objeto o_i y ii) decrementar el valor de aquellos atributos que no estén en ambos. Para aumentar o decrementar el nivel de confianza relativo a un atributo del concepto almacenado en el nodo, se modifica el vector numérico almacenado en dicho nodo en la dimension que representa el valor de confianza del atributo en cuestión. Por lo general, el valor a aumentar o decrementar para un atributo está relacionado con la distancia que existe entre el concepto del nodo y el objeto, en términos de dicho atributo. Una vez evaluado un nodo, UNIMEN puede eliminar un atributo de su concepto si su valor de confianza es menor a un umbral p_4 o volverlo fijo (no modificar más su confianza) si su valor es superior a un umbral p_5. Eventualmente, producto del proceso de evaluación un concepto puede perder muchos atributos y volverse tan general que no aporta conocimiento. En tal caso, UNIMEN opta por eliminar dicho nodo y re-procesar los objetos incluidos en algún nodo del sub-árbol definido por el nodo eliminado, que no están en algún otro nodo de la jerarquía.

Entre las limitaciones que tienen UNIMEN y RESEARCHER se encuentra el número de parámetros que necesitan ajustar, lo cual hace difícil la aplicación de estos algoritmos en problemas reales. Otra limitación es que la estructura obtenida depende del orden de análisis de los objetos y por lo tanto, pueden obtenerse diferentes jerarquías a partir de una misma colección, dependiendo del orden en que se analicen sus objetos. Más aún, el problema de la dependencia del orden puede afectar la calidad de los conceptos generados, pues algunos conceptos pudieran fijar algunos atributos dependiendo del orden de análisis y no de las características reales de la colección. Otra limitación de estos algoritmos, sobre todo de RESEARCHER, es el considerable gasto de memoria que tienen, lo cual restringe el número de objetos que pueden procesar y dificulta su uso en problemas reales. Por último, estos algoritmos no pueden procesar objetos descritos por rasgos incompletos.

2.5. OPUS

OPUS [25] es un algoritmo de agrupamiento que permite formar una jerarquía de conceptos. Este algoritmo asume que los objetos están descritos por atributos cualitativos multievaluados (i.e., conjuntos de valores) y que se conoce, como entrada del algoritmo, un conjunto de relaciones binarias que son satisfechas por los objetos a agrupar.

OPUS considera como nodo raíz de la jerarquía al grupo que contiene a todos los objetos de la colección. A partir de este punto, OPUS forma la jerarquía utilizando un enfoque divisivo, de forma que los grupos formados son disjuntos. Como parte de este enfoque, este algoritmo recorre cada nodo de la jerarquía, particionando el conjunto de objetos almacenados en dicho nodo. Para obtener esta partición, OPUS emplea una estrategia compuesta de dos pasos. Sea N el nodo a procesar. En el primer paso, se evalúa la calidad de la partición que se puede formar si se fija un atributo r_j y se particiona el conjunto de objetos contenido en N, de acuerdo a los diferentes valores que toman los objetos del nodo para dicho atributo. Sea P_{r_j} la partición que se obtiene a partir del atributo r_j. Para evaluar la calidad de P_{r_j}, denotada por $Q(P_{r_j})$, primeramente se forma para cada grupo de la partición, el l-complejo que lo describe, considerando el resto de los atributos que describen a los objetos. Sea $L_{r_j} = \{l_1, l_2, \ldots, l_m\}$ los l-complejos que describen a P_{r_j}. La fórmula para calcular $Q(P_{r_j})$ es la siguiente:

$$Q(P_{r_j}) = \alpha \cdot SPL(P_{r_j}) + \beta \cdot DFI(P_{r_j}),$$

donde $SPL(P_{r_j})$ denota la simplicidad total de los l-complejos que describen a los grupos de P_{r_j} y $DFI(P_{r_j})$ denota la diferencia inter-grupo promedio, que existe entre los grupos de P_{r_j}.

Para calcular $SPL(P_{r_j})$, primero se calcula la complejidad de cada selector incluido en cada l-complejo $l_i \in L_{r_j}$ y con base en esto, se calcula la complejidad de L_{r_j}; $SPL(P_{r_j})$ se calcula como el opuesto de la complejidad de L_{r_j}. La complejidad de un selector se calcula como la razón entre el número de valores que toma el selector y el número de elementos en el dominio de valores del atributo que describe dicho selector. La complejidad de L_{r_j} se calcula como el promedio de complejidad de todos los selectores incluidos en los l-complejos de L_{r_j}.

Como se mencionó anteriormente, OPUS asume que los valores de los atributos son conjuntos de valores. Con base en esto, la semejanza entre dos valores e_1, e_2 de un selector se calcula utilizando el índice Jaccard [26]; es decir, se calcula como $\frac{|e_1 \cap e_2|}{|e_1 \cup e_2|}$. La semejanza de un valor e_1 de un selector, respecto a otro selector s_1 que describe al mismo atributo, se calcula como el máximo de las semejanzas que existen entre e_1 y cada valor e de s_1. La semejanza entre un selector s_1 respecto a un selector s_2, ambos describiendo al mismo atributo, se calcula como el promedio de las semejanzas que existe entre los valores de s_1 respecto a los valores del selector s_2. Utilizando la fórmula anterior, la semejanza de un l-complejo l_1 respecto a otro l-complejo l_2, denotado como $Sim(l_1, l_2)$, se define como el promedio de las semejanzas que existen entre los selectores de l_1 y l_2 que describen a un mismo atributo. Finalmente, el valor de $DFI(P_{r_j})$ se calcula de la siguiente forma:

$$DFI(P_{r_j}) = 1 - \frac{\sum_{l_i, l_j \in L, l_i \neq l_j} Sim(l_i, l_j)}{|L| \cdot (|L| - 1)}.$$

Una vez calculada $Q(P_{r_j})$, para cada atributo r_j que describe a los objetos, en el segundo paso OPUS selecciona el atributo que forma la partición de mayor calidad y particiona el conjunto de objetos contenidos en N de acuerdo a dicho atributo. Cada grupo formado se adiciona a la jerarquía como hijo del nodo N. El concepto que describe intencionalmente a un grupo N' hijo del nodo N, se forma con la concatenación de los conceptos que describen a los nodos que están en el camino desde la raíz hasta M, más el valor del atributo seleccionado que forma a N', más todos aquellos valores del resto de los atributos, que sean compartidos por todos los objetos contenidos en N'. El proceso anterior se repite en cada nodo descendiente de N, utilizando atributos que no hayan sido seleccionados previamente en esa rama. Cuando utilizando el conjunto de atributos restante ya no se pueda particionar más el conjunto de objetos contenidos en un nodo, entonces OPUS utiliza las relaciones existentes entre los objetos, así como los atributos que describen a éstos, para generar nuevos atributos. Utilizando estos nuevos atributos, OPUS intenta seguir particionando el conjunto de objetos contenidos en los nodos, utilizando la estrategia descrita hasta el momento. Cada vez que no se pueda seguir particionando los nodos con los atributos actuales se sigue el mismo proceder. Esto es, se generan nuevos atributos y se intenta seguir particionando con base en estos atributos y la estrategia de particionado descrita anteriormente. El proceso de formación de la jerarquía se detiene cuando no se puedan seguir generando nuevos atributos o cuando, al generar nuevos atributos, no se pueda particionar el conjunto de objetos del nodo.

Para crear nuevos atributos se utilizan las relaciones existentes entre los objetos, que fueran entradas al algoritmo como parámetros, así como los atributos que describen a los objetos. El conjunto de relaciones dadas como parámetros y el inverso de dichas relaciones, constituyen las relaciones *primitivas*. Para cada camino desde la raíz a los nodos, se almacena el número de veces que ha sido necesario crear nuevos atributos para poder particionar el conjunto de objetos de un nodo en dicho camino. Sea K_N dicho número para el nodo N de la jerarquía. Para crear nuevos atributos que permitan particionar el conjunto de objetos del nodo N, se necesita crear relaciones complejas de orden $K_N + 1$. Una relación compleja de orden K_N, se denota como $r.f(X, Y, Z)$ y se compone de una relación $r(X, Y)$ de *orden* K_N entre los objetos X, Y y del

atributo $f(Y,Z)$ perteneciente al objeto Y. El orden de una relación es el número de relaciones primitivas que la conforman. De esta forma, una relación de orden 1 es una relación primitiva. Una relación de orden $K_N > 1$ se obtiene a través de la combinación de K_N relaciones primitivas. Por ejemplo, en el contexto de la cadena alimenticia, una relación primitiva puede ser $Eat(X,Y)$ que establece que el animal X se come al animal Y. En este mismo contexto, un ejemplo de relación de orden 2 es la relación $Eat - Eat(X,Z)$ que establece que un animal X se come a un animal Y, el cuál a su vez, se come a un animal Z.

Los nuevos atributos para un objeto X se forman con los valores de los atributos Z, en las relaciones complejas $r.f(X,Y,Z)$ de orden K_N, que hacen que la relación $r(X,Y)$ de orden K_N se cumpla para algún objeto Y. Por ejemplo, suponga la relación compleja $Eat - Size(X,Y,Z)$ de orden 1, que establece que un animal X se come a un animal Y, que tiene un tamaño igual al valor Z. Con base en esta relación, el valor del nuevo atributo para el objeto $X =$"serpiente", son los valores del atributo "Size" que hacen posible que la relación de orden 1 $Eat($"serpiente",$Y)$ se cumpla para algún objeto Y de la colección.

Entre las limitaciones del algoritmo se encuentra que puede construir conceptos muy grandes y por lo tanto, difíciles de interpretar en problemas reales. Adicionalmente, OPUS necesita contar con un conjunto de relaciones entre los objetos y este conocimiento previo no se conoce siempre en todos los problemas reales. Por último, la complejidad computacional de OPUS puede ser alta, en dependencia del número de relaciones complejas que se tienen que crear y verificar, así como del número de veces que se necesita crear nuevos atributos.

2.6. COBWEB y sus variantes

En esta sección se describe el algoritmo de agrupamiento conceptual COBWEB [23], así como los algoritmos CLASSIT [27], COBWEB/3 [28], CLASSIT+ [29], DynamicWEB [30,31], ARACHNE [32], COBBIT [33], BRIDGER [34], INC [35] y HGA-COBWEB [36], los cuales son extensiones de COBWEB. Todos estos algoritmos permiten formar jerarquías de grupos disjuntos.

2.6.1. COBWEB

COBWEB [23] es un algoritmo incremental que permite construir una jerarquía de conceptos. En esta jerarquía se busca que los grupos formados en cada nivel, maximicen la información que se puede predecir, a partir de conocerse la pertenencia de un objeto al mismo. COBWEB asume que los objetos de la colección están descritos como pares atributo-valor y que los atributos son cualitativos. Los conceptos que describen a cada grupo (nodo de la jerarquía) se forman con los atributos que describen a los objetos de la colección y expresan la probabilidad de ocurrencia de cada par atributo-valor del concepto; esta probabilidad se calcula a partir de los objetos almacenados en el sub-árbol definido por el nodo.

Para formar una jerarquía de conceptos, COBWEB parte de una jerarquía vacía y procesa los objetos de la colección de uno en uno. Sea o_i el objeto que se desea adicionar a la jerarquía. Para encontrar el nodo que contendrá a o_i se realiza una búsqueda recursiva, que parte desde el nodo raíz de la jerarquía y en la cual se realizan varios pasos sobre cada nodo recorrido. En el primer paso se verifica si el nodo actual es terminal y si lo es, entonces se crea un nodo T que tendrá como nodos hijos al nodo actual y al objeto o_i. El concepto del nodo T se construye a partir del concepto del nodo actual y de los valores de los atributos de o_i. Posteriormente, el nodo T se inserta en la jerarquía en el lugar que ocupaba el nodo actual. En otro caso, si el nodo actual no es terminal, entonces se actualiza el concepto almacenado en el nodo actual, de acuerdo a las características de o_i y a continuación, se sigue un proceso compuesto de cinco pasos.

En el primer paso de este proceso, se calcula la calidad de las particiones que se obtienen al colocar a o_i en cada nodo hijo del nodo actual y considerar el resto de los nodos como tal. Sea P_1 la partición que

alcanza el mayor valor de calidad en el paso anterior. En el segundo paso, se determina la calidad de la partición P_2, obtenida al considerar los nodos hijos como tal y a o_i como un nuevo nodo hijo. En el tercer paso, se calcula la calidad de la partición que se forma al unir los dos nodos hijos que formaban las dos mejores particiones en el paso 1 y considerar al resto de los grupos como tal; sea esta partición P_3. En el cuarto paso, se determina la calidad que se forma si se elimina el nodo hijo que forma la partición P_1 y en su lugar se consideran a sus hijos; sea esta partición P_4. Por último, en el quinto paso se comparan las calidades de las cuatro particiones analizadas en los pasos anteriores. Si la mejor es P_1, entonces se considera como nodo actual al nodo que forma la partición P_1 y se continúa la búsqueda a partir de dicho nodo. Si la mejor es P_2, entonces se crea un nuevo nodo T que contiene al objeto o_i y se adiciona T a la jerarquía como hijo del nodo actual; el concepto que describe a T se construye a partir de las propiedades de o_i. Si la mejor es P_3, entonces se crea un nuevo nodo T que tiene como hijos a los nodos que forman la partición P_3, se eliminan dichos nodos de la jerarquía y se adiciona el nodo T como hijo del nodo actual; el concepto que describe a T se forma a partir de las características de sus nodos hijos. A continuación, el nodo T se considera como nodo actual y se sigue la búsqueda a partir del mismo. Por último, si la mejor partición es P_4, entonces se elimina el nodo que permite formar P_4 y sus hijos son adicionados como hijos del nodo actual. Hecha esta operación, el proceso de búsqueda se vuelve a ejecutar sobre el nodo actual.

Para medir la calidad de una partición, COBWEB utiliza una medida llamada Utilidad de Categoría (CU, por sus siglas en inglés). Esta medida obtiene valores altos para aquellas particiones en las cuales haya una alta semejanza entre los objetos de un mismo grupo y una baja semejanza entre objetos de grupos diferentes. La medida CU se define como sigue:

$$CU = \frac{\sum_{k=1}^{N} \left(P(C_k) \cdot \left[\sum_i \sum_j P(A_i = V_{ij} \mid C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2 \right] \right)}{N},$$

donde $P(C_k)$ es la probabilidad de ocurrencia del grupo C_k, $P(A_i = V_{ij} \mid C_k)$ es la probabilidad de que el atributo A_i del objeto tenga valor V_{ij}, dado que se conoce que el objeto pertenece al grupo C_k, $P(A_i = V_{ij})$ es la probabilidad de que el atributo A_i del objeto tenga valor V_{ij}.

COBWEB es un algoritmo que construye al unísono la descripción intencional y extensional de los grupos. Entre las limitaciones de este algoritmo está su alta complejidad computacional, debido a las varias opciones que hay que realizar durante el proceso de búsqueda sobre la jerarquía. Otra limitación es que solo puede procesar objetos descritos por atributos cualitativos. Por otro lado, aunque COBWEB es un algoritmo incremental, solo puede incorporar los objetos en la jerarquía de uno en uno, por lo que en contextos donde los datos se adicionan frecuentemente COBWEB puede resultar poco eficiente. Si bien es cierto que, los conceptos generados por COBWEB tienen un fundamento más sólido al estar basados en probabilidades, también resultan poco intuitivos y en la práctica son difíciles de interpretar para los usuarios finales que generalmente no saben de probabilidades o estadísticas. Por último, aunque COBWEB permite modificar la estructura de la jerarquía, a través de la unión o eliminación de nodos, este algoritmo es dependiente del orden de análisis de los objetos, por lo que diferentes ordenes de una misma colección de objetos pueden formar diferentes jerarquías.

2.6.2. CLASSIT, CLASSIT+ y COBWEB/3

CLASSIT [27] es un algoritmo que extiende COBWEB, haciéndolo capaz de procesar objetos descritos por atributos numéricos. Para esto CLASSIT asume que los valores de los atributos de los objetos siguen una distribución normal y por lo tanto, modifica la forma en que se construyen los conceptos que describen a los grupos, así como la forma de calcular la medida CU. Ahora los conceptos no contienen la probabilidad de ocurrencia que tienen los pares atributo-valor, sino que contienen la media y la desviación estándar de los valores de cada atributo. Para construir la jerarquía de conceptos este algoritmo mantiene

la misma estrategia de COBWEB pero modifica la medida CU, quedando esta de la siguiente forma:

$$CU = \frac{\sum_{k=1}^{N} \left(P(C_k) \cdot \left[\sum_i \frac{1}{\sigma_{ik}} - \sum_i \frac{1}{\sigma_{ip}} \right] \right)}{N},$$

donde N es el número de grupos de la partición, σ_{ik} y σ_{ip} son las desviaciones estándar del valor del atributo i-ésimo en el grupo k y en l nodo padre del grupo k. Es importante notar que esta fórmula se indefine cuando el concepto solo cuenta con un solo objeto, pues en este caso la desviación sería cero. Para evitar la indefinición en estos casos, CLASSIT utiliza un valor mínimo de desviación que es especificado como parámetro al algoritmo.

Además del cambio en la definición de la medida CU, CLASSIT introduce un parámetro llamado "corte" que le permite tener que descender siempre hasta las hojas de la jerarquía de conceptos, cuando se va a clasificar a un objeto. Cuando el valor de la medida CU en un nodo sobrepasa el valor del parámetro "corte" se dice que el objeto es lo suficiente semejante al concepto almacenado en el nodo actual y por lo tanto, carece de sentido seguir descendiendo en la jerarquía.

Por otra parte, CLASSIT+ [29] es una extensión del algoritmo CLASSIT. En esta extensión, se asume que los valores de los atributos no siguen una distribución normal sino que siguen una distribución de Katz [37] y por lo tanto, se redefine la fórmula de la medida CU.

COBWEB/3 [28] es una extensión de COBWEB que combina las metodologías aplicadas en COBWEB y en CLASSIT, permitiendo ser capaz de procesar objetos descritos por atributos mezclados. Para esto, COBWEB/3 modifica la definición de la medida CU, de forma que ésta se calcula como la suma de la medida CU definida por COBWEB para los atributos cualitativos, más la medida CU definida por CLASSIT para los atributos numéricos.

De forma similar a COBWEB, la limitación principal de estos algoritmos es la alta complejidad computacional que tienen, producto del mismo procedimiento de inserción del objeto en la jerarquía. Adicionalmente, al igual que COBWEB, estos tres algoritmos solo pueden incorporar los objetos en la jerarquía de uno en uno, por lo que en contextos donde los datos se adicionan frecuentemente COBWEB pueden resultar poco eficientes. Una limitación de CLASSIT y CLASSIT+ es que solo pueden procesar objetos descritos por atributos numéricos. Por otra parte, COBWEB/3 no puede procesar objetos con atributos incompletos. Por último, los tres algoritmos son dependientes del orden de análisis de los objetos.

2.6.3. DynamicWEB

DynamicWEB [30,31] es una extensión de COBWEB capaz de procesar eliminaciones y modificaciones de objetos; las modificaciones son tratadas como una eliminación seguida de una adición. Este algoritmo utiliza la misma estrategia de COBWEB para procesar las adiciones de objetos. No obstante, a diferencia de COBWEB, DynamicWEB mantiene al unísono un *árbol AVL* [38] en el cual indexa a todos los objetos de la colección, de conjunto con la referencia al lugar de la jerarquía de conceptos en donde se encuentran. Un árbol AVL es un árbol binario de búsqueda que está balanceado por la altura.

Cuando se desea eliminar un objeto almacenado, éste se localiza rápidamente a través del árbol AVL y es eliminado tanto de la jerarquía como del árbol AVL. Consecuentemente, el árbol AVL es actualizado utilizando el procedimiento definido para esta estructura de datos. De igual forma, la jerarquía de conceptos es actualizada, siguiendo una estrategia compuesta de cinco pasos. En el primer paso, se actualizan los valores de probabilidad del concepto que describe al nodo del cual fue eliminado el objeto; sea este nodo N. Posteriormente, en el segundo paso se verifica si N es vacío. Si este es el caso, entonces se elimina N y se repite el proceso anterior desde el primer paso, en el nodo padre de N. En otro caso, si N no es vacío, entonces en el tercer paso se utiliza al objeto eliminado para determinar cuáles son los nodos hijos N_1 y

N_2 de N que, después de asimilar al objeto N en su descripción extensional, forman las dos particiones de mayor calidad, de acuerdo a la medida CU. Posteriormente, en el cuarto paso se determina el valor de la medida CU, de la partición resultante de unir los nodos N_1 y N_2, y el valor de la medida CU, de la partición resultante de eliminar a N_1 y poner a sus hijos como hijos del nodo padre de N_1; sean estas particiones P_1 y P_2, respectivamente. En el quinto paso, si el valor de CU de P_1 es mayor que un parámetro *cutoff*, entonces se considera unir los nodos N_1 y N_2, de acuerdo a la estrategia propuesta originalmente por COBWEB. En otro caso, se verifica si el valor de CU de P_2 es mayor que un parámetro *cutoff* y si es así, entonces se considera eliminar a N_1 y poner a los hijos de éste como hijos del nodo N, tal como establece COBWEB. Si alguna de las dos posibilidades anteriores se llegó a realizar, entonces se repite el proceso anterior desde el primer paso, en el nodo padre de N. En otro caso, si los valores de CU de P_1 y P_1 son ambos menos que *cutoff*, entonces no se realiza cambio alguno y solo se continúa actualizando el valor de probabilidad de los conceptos almacenados en el camino desde el padre de N hasta la raíz.

Cuando se modifica un objeto de la colección, éste se procesa siguiendo la estrategia comentada anteriormente y luego, es adicionado con los valores de los atributos actualizados (i.e., modificado) y procesado según la estrategia básica de COBWEB.

DynamicWEB presenta las mismas limitaciones que COBWEB y además, incluye una nueva limitación: la alta complejidad computacional del proceso de actualización de la jerarquía, luego de eliminaciones y modificaciones.

2.6.4. ARACHNE

ARACHNE [32] es un algoritmo de agrupamiento conceptual que, utilizando una estrategia muy parecida a la de COBWEB, organiza una colección de objetos como una jerarquía de conceptos. No obstante, ARACHNE extiende la estructura de control de COBWEB haciéndolo menos dependiente del orden de análisis los objetos.

Para incorporar un objeto a la jerarquía ARACHNE emplea la misma estrategia que COBWEB. No obstante, una vez que el objeto es insertado en un nodo N, ARACHNE verifica el cumplimiento de dos restricciones en todos los nodos hijos y hermanos de N. Es importante mencionar que para la verificación de estas restricciones ARACHNE asume la existencia de una medida que permite calcular la semejanza entre un par de objetos, entre un par de conceptos o entre un objeto y un concepto.

La primera restricción verifica que todos los nodos hijos de N estén bien posicionados verticalmente. Para esto, se calcula para cada uno de estos nodos hijos, la semejanza que existe entre él y N y la semejanza entre él y el nodo padre de N (su nodo *abuelo*). Si un nodo X es más semejante a su nodo abuelo Y que a su nodo padre N, entonces X es eliminado de la lista de hijos de N y es adicionado como hijo de Y. Es importante mencionar que a la vez que se hace esta operación, también se actualiza el concepto almacenado en el nodo N. Una vez que se verifica la primera restricción en todos los nodos hijos de N, ARACHNE verifica que todos los nodos hermanos de N estén bien colocados horizontalmente. Sea Y el nodo padre de N. Para verificar esta segunda restricción, ARACHNE calcula la semejanza que existe entre todos los pares de nodos hijos de Y y determina si hay algún par de nodos que sean más semejantes entre sí, que lo que lo son cada uno a Y. Si existe algún par que cumpla esta condición, entonces ARACHNE mezcla estos nodos, de la misma forma en que lo hace COBWEB. Esta condición se sigue verificando iterativamente hasta que todos los nodos hijos de Y la cumplan.

De forma similar a COBWEB, ARACHNE solo puede procesar objetos descritos por atributos cualitativos y forma conceptos que en la practica resultan poco intuitivos y difíciles de interpretar para los usuarios finales que generalmente no saben de probabilidades o estadísticas. Además, ARACHNE también está obligado a incorporar los objetos en la jerarquía de uno en uno, por lo que en contextos donde los datos se adicionan frecuentemente puede ser poco útil. Hay que mencionar además que, si bien el proceso

de verificación de las dos restricciones propuestas permite reducir la dependencia del orden de análisis de los objetos y formar mejores jerarquías, también aumenta la complejidad computacional de ARACHNE.

2.6.5. COBBIT y BRIDGER

COBBIT [33] es un algoritmo que extiende COBWEB con el objetivo de atacar tres puntos: a) los problemas de escalabilidad que presenta COBWEB con colecciones de muchos objetos, b) el mantenimiento de conceptos que luego de algún tiempo se vuelvan *obsoletos* y c) la dependencia del orden de análisis de los objetos de la que padece COBWEB.

Para tratar estos problemas COBBIT propone el uso de un ventana de tiempo, con un tamaño definido por un parámetro, que funciona como una estructura FIFO (First In, First Out) en la cual, el primer objeto en entrar es el primero en salir. De esta forma, cuando se adiciona un objeto este se inserta en la jerarquía siguiendo la misma estrategia definida por COBWEB y, si el número de objetos almacenados supera el tamaño de la ventana de tiempo, entonces el objeto más *antiguo* (el del principio de la estructura) es eliminado de la jerarquía. Una vez que se elimina este objeto, los conceptos de los nodos afectados son actualizados, considerando los objetos que actualmente están en el sub-árbol definido por cada nodo. Si producto de esta eliminación algún nodo queda vacío, éste es eliminado de la jerarquía.

BRIDGER [34] es una extensión de COBWEB que permite procesar objetos descritos por atributos mezclados. Para esto, BRIDGER utiliza la definición de CU que emplea COBWEB/3. Para formar una jerarquía de conceptos, BRIDGER utiliza la misma estructura de control que COBWEB. No obstante, BRIDGER extiende esta estructura de control para que, además de guiar la búsqueda del mejor nodo para insertar al objeto, permita generar nuevos atributos si esta acción mejora el CU de la partición que se está considerando. BRIDGER considera dos tipos de acciones para generar nuevos atributos. La primera es crear atributos que tengan múltiples valores y la segunda es dividir un atributo en varios, de acuerdo al dominio de valores del mismo. Para evitar probar todas las combinaciones posibles, BRIDGER solo trata de crear nuevos atributos a partir de los atributos presentes en el objeto que se procesa en cada momento

Entre las limitaciones de COBBIT se encuentra que, de forma similar a COBWEB, solo trabaja con objetos descritos por rasgos cualitativos. Adicionalmente, COBBIT puede formar conceptos que en la practica resultan poco intuitivos y difíciles de interpretar. Por otra parte, COBBIT también está obligado a incorporar los objetos en la jerarquía de uno en uno, por lo que es poco útil en contextos donde los datos se adicionan frecuentemente. Por otra parte, BRIDGER mantiene también varias de las limitaciones de su predecesor COBWEB y adiciona la alta complejidad computacional del proceso de creación de nuevos atributos; este proceso afecta considerablemente la aplicación de BRIDGER en problemas reales con colecciones de muchos objetos.

2.6.6. INC

INC [35] es un algoritmo incremental que construye una jerarquía de conceptos, empleando una estrategia similar a COBWEB. INC asume que los objetos se describen por pares atributo-valor, en el cual los atributos son cualitativos. No obstante, a diferencia de COBWEB, INC asume también que cada par atributo-valor que describe a un objeto tiene asociado un valor numérico que expresa la relevancia de dicho par en la descripción del objeto. De forma similar, INC describe a los conceptos que forma por pares atributo-valor y almacena para cada par un valor numérico que representa la relevancia de dicho par atributo-valor para describir al concepto. La relevancia que tiene un par atributo-valor para describir a un concepto se calcula como el promedio de relevancia que tiene dicho par en las descripciones de los objetos que están almacenados en dicho concepto.

Para procesar la adición de un objeto, INC no utiliza la medida CU sino que utiliza una medida de semejanza que determina el *grado de membresía* de dicho objeto a un concepto determinado. Sea o un objeto que INC debe procesar y c un concepto que describe a los objetos almacenados en un nodo de la jerarquía formada. Sea M el conjunto de atributos que tienen tanto en c como en o el mismo valor. El grado de membresía del objeto o en el concepto c se denota por $M(c,o)$ y se calcula como sigue:

$$M(c,o) = \frac{\sum_{att_i \in M} min\{rel(c,att_i,val), rel(o,att_i,val)\}}{|M|},$$

donde $rel(c,att_i,val)$ y $rel(o,att_i,val)$ representan la relevancia que tiene el par (att_i,val) en el concepto c y el objeto o, respectivamente.

En este proceso de búsqueda, el primer paso es actualizar el concepto del nodo actual, utilizando los pares atributo-valor del objeto en proceso. Esta actualización incluye además la modificación de los valores de relevancia para los pares atributo-valor que describen al concepto, así como la eliminación de cualquier par cuya relevancia caiga por debajo de un umbral predefinido. Posteriormente, en el segundo paso se calcula la semejanza entre el objeto y todos los nodos hijos del nodo actual, seleccionando aquél que maximice esta semejanza, siempre y cuando ésta sea superior a 0.5. Si existe dicho nodo hijo, el proceso de búsqueda continúa a través de él. En otro caso, si más de un nodo alcanza el máximo de semejanza con el objeto, siendo ésta superior a 0.5, INC crea un nuevo nodo T que contiene como hijos al objeto adicionado y a los nodos hijos de N que alcanzan dicho máximo de semejanza. Posteriormente, estos nodos son eliminados de la lista de nodos hijos del nodo actual y en su lugar se inserta el nodo T. Por el contrario, si la semejanza máxima fue inferior a 0.5, entonces se selecciona el nodo P hijo del nodo actual que alcanza dicho máximo y se comprueba la semejanza del objeto con los nodos hijos de P. Si ningún nodo hijo de P alcanza una semejanza con el objeto, mayor a 0.5, entonces se crea un nuevo nodo que contiene al objeto adicionado y se inserta dicho nodo como hijo del nodo actual. En otro caso, si al menos un nodo P', hijo de P, tiene una semejanza mayor a 0.5 con el objeto, entonces P se elimina de la lista de hijos del nodo actual y en su lugar se insertan dos nodos T y T'. El nodo T contiene como hijos al objeto y al nodo P'. Por otra parte, el nodo T' contiene al resto de los nodos hijos de P.

Entre las limitaciones de INC se encuentra que solo procesa objetos descritos por atributos cualitativos. Otra limitación es que, de forma similar a COBWEB, INC puede formar conceptos que en la practica resultan poco intuitivos y difíciles de interpretar. Además, INC está obligado a incorporar los objetos en la jerarquía de uno en uno, por lo que es poco útil en contextos donde los datos se adicionan frecuentemente. Por último, este algoritmo requiere ajustar valores para tres parámetros que dependen de la colección a procesar, lo cual dificulta su aplicación en problemas reales.

2.6.7. HGA-COBWEB

HGA-COBWEB [36] es un algoritmo de agrupamiento conceptual, que permite formar una jerarquía de conceptos y actualizar dicha jerarquía cuando la colección cambia. Este algoritmo es una extensión del algoritmo COBWEB/3 y por lo tanto, asume que los objetos pueden estar descritos por atributos numéricos y cualitativos. No obstante, a diferencia de COBWEB y el resto de las variantes antes comentadas, HGA-COBWEB puede procesar múltiples adiciones de objetos.

Para formar la jerarquía de conceptos, HGA-COBWEB emplea dos etapas. El objetivo de la primera etapa es mitigar el efecto del orden de análisis de los objetos en la formación de la jerarquía. Los autores HGA-COBWEB asumen que esta dependencia del orden se acentúa cuando los objetos son incorporados uno auno a la jerarquia y por lo tanto, proponen en esta primera etapa particionar el conjunto de objetos que se tiene como entrada del algoritmo. Por otra parte, la segunda etapa es la encargada de procesar cada grupo de la partición formada en la primera etapa e incorporarlo a la jerarquía de conceptos.

Para formar la partición de objetos en la primera etapa, se construye una jerarquía utilizando un enfoque divisivo. Este enfoque parte de considerar a todos los objetos como el nodo raíz de la jerarquía y recursivamente divide en dos el conjunto de objetos almacenados en cada nodo, hasta alcanzar la condición de parada. Para dividir en dos el conjunto de objetos contenidos en un nodo, se utiliza un algoritmo genético que emplea la medida CU definida por COBWEB, como función de aptitud. El proceso de construcción de esta jerarquía se detiene cuando se construyen cuatro niveles. En este contexto, el conjunto de objetos contenidos en cada nodo terminal de esta jerarquía constituye un grupo de la partición determinada en la primera etapa. Es importante mencionar que en ningún momento se brinda justificación alguna para detener la construcción de la jerarquía en el cuarto nivel, aunque puede ser para disminuir el costo computacional de esta primera etapa. Posteriormente, en la segunda etapa cada grupo de esta partición es insertado en la jerarquía de conceptos utilizando una variante de COBWEB/3, modificada para procesar conjuntos de objetos en vez de un solo objeto a la vez.

De forma similar a COBWEB y COBWEB/3, una de las limitaciones de HGA-COBWEB es su alta complejidad computacional, la cual aumenta también debido al proceso efectuado en la primera etapa. Adicionalmente, el algoritmo genético que se utiliza en la primera etapa requiere de varios parámetros que dependen de la colección a procesar, lo cual puede dificultar la aplicación de HGA-COBWEB en problemas reales. Por último, este algoritmo no puede procesar objetos descritos por atributos incompletos.

2.7. ITERATE

ITERATE [39,40] es un algoritmo conceptual que combina el paradigma jerárquico y particional, con el objetivo de generar un conjunto de grupos disjuntos que son cohesionados y diferentes entre sí. Para obtener dicho agrupamiento, este algoritmo reutiliza algunos conceptos previamente introducidos por COBWEB [23].

Para generar el agrupamiento y la descripción de cada grupo, el algoritmo ITERATE emplea tres etapas. En la primera etapa, ITERATE construye una jerarquía de conceptos con los objetos de la colección en dicho árbol. Posteriormente, en la segunda etapa se extrae una partición inicial de dicha jerarquía de conceptos y por último, en la tercera etapa se realiza un proceso de redistribución de los objetos, hasta formar grupos máximamente separables.

Para la construcción de la jerarquía de conceptos en la primera etapa, ITERATE sigue una estrategia similar a la de COBWEB, lo que en este caso dicha jerarquía se construye a lo ancho; es decir, cada nivel es completado antes de crear el nivel siguiente. Esta estrategia parte de considerar a todos los objetos de la colección como parte del nodo raíz e iterativamente divide el conjunto de objetos de cada nodo en subclases. Para particionar el conjunto de objetos almacenados en un nodo, se emplean dos pasos. En el primer paso, se selecciona del conjunto de objetos almacenados en el nodo, aquel que minimice la suma de las distancias de Manhattan desde él a los n primeros objetos seleccionados; donde n es un parámetro del algoritmo. El primer objeto en ser seleccionado es aquel que tenga la mayor semejanza con el centroide del nodo. Posteriormente, en el segundo paso se determina si el objeto o seleccionado se adiciona a alguno de los grupos creados hasta ese momento o si forma uno nuevo. Para este propósito, primero se calcula la calidad de cada una de las particiones resultantes de adicionar a o a uno de los grupos y tomar el resto en su forma original; para calcular la calidad de cada partición se utiliza la medida CU empleada por el algoritmo COBWEB (ver subsección 2.6.1). Una vez determinada la calidad de cada partición, se selecciona la de mejor calidad y se compara ésta con la calidad de la partición resultante de crear un nuevo grupo con o. Si crear un nuevo grupo lleva a una partición de mejor calidad, entonces se crea un nuevo grupo que contiene solo a o; en otro caso, se adiciona el objeto o a la mejor partición determinada. Es importante mencionar que, cada vez que se crea un nuevo grupo o se actualiza un grupo existente por la inserción de un nuevo

objeto, se actualiza el concepto probabilístico que describe a los objetos almacenados en dicho nodo, de la misma forma en que lo hace COBWEB. Los pasos uno y dos se repiten hasta que se procesan todos los objetos almacenados en el nodo.

Es importante mencionar que, aunque la versión básica de ITERATE solo permite procesar objetos descritos por rasgos cualitativos, se han explorado algunas variantes para permitir el procesamiento de objetos descritos por atributos numéricos o mezclados. Las variantes que se han explorado consisten en: i) utilizar funciones de densidad de probabilidad en vez de probabilidades, de forma similar al algoritmo CLASSIT [27] o ii) discretizar los valores numéricos [39]. De estas dos variantes, la que los autores reportan que les ha funcionado mejor es la de discretizar los atributos numéricos.

Una vez formado el árbol de clasificación, en la segunda etapa se construye una partición inicial a partir de este árbol. Para este propósito se recorre el árbol en profundidad, desde la raíz. Durante este recorrido, si el nodo actual tiene un valor de CU mayor que el de todos sus nodos hijos, entonces el grupo de objetos contenidos en el nodo actual constituye la descripción extensional de un grupo en la partición inicial, deteniéndose el recorrido por esta vía. Por el contrario, el recorrido sigue por aquellos nodos hijos que tengan mayor valor de CU que el nodo actual. Adicionalmente, si existieran nodos hijos con menor valor de CU que el nodo actual, los objetos contenidos en cada uno de esos nodos hijos constituyen también la descripción extensional de un grupo en la partición inicial. Es importante mencionar que cada vez que se crea un grupo con los objetos contenidos en un nodo, la descripción intencional de dicho grupo la constituye el concepto probabilístico que describe a los objetos de dicho nodo. Esta forma de calcular la partición inicial permite formar un agrupamiento que tiene la mayor calidad y en el cual ningún grupo o concepto incluye a ningún otro.

Sea $G = \{G_1, G_2, \ldots, G_k\}$ la partición inicial formada en la segunda etapa. En la tercera etapa se realiza un proceso iterativo, en el cual los objetos son redistribuidos en los grupos, con el objetivo de maximizar la cohesión de los mismos. En cada iteración se recorren todos los objetos de la colección y para cada objeto $o_i \in O$, se calcula el valor de la medida CM existente entre o_i y cada grupo de G. Posteriormente, se asigna el objeto o_i al grupo con el cual éste alcanzó el mayor valor de la medida CM. En caso de existir empate, si el grupo al cual pertenece o_i está entre aquellos con los cuales o_i alcanza el mayor valor de CM, entonces o_i se queda en su grupo. En otro caso, si el grupo al cual pertenece o_i no está entre los mejores de acuerdo a la medida CM, entonces el empate se rompe arbitrariamente. Es importante mencionar que cada vez que un objeto cambia de grupo, se actualiza el concepto probabilístico que describe al grupo origen y al grupo destino. La medida CM entre un objeto $o_i \in O$ y un grupo $G_j \in G$ se denota por $CM_{i,j}$ y se calcula como sigue:

$$CM_{i,j} = P(G_j) \cdot \sum_{r_q \in o_i} \sum_{v_t \in D_q} \left(P(r_q = v_t \mid G_j)^2 - P(r_q = v_t)^2 \right),$$

donde $P(G_j)$ es la probabilidad del grupo G_j, r_q es un atributo que describe al objeto o_i, D_q es el dominio de valores del atributo r_q, $P(r_q = v_t)$ es la probabilidad de que el atributo r_q que describe al objeto o_i alcance el valor v_t y $P(r_q = v_t \mid G_j)$ es la probabilidad de que el atributo r_q que describe al objeto o_i alcance el valor v_t, dado que se conoce que el objeto o_i está en el grupo G_j. El proceso de redistribución continúa hasta que ningún objeto cambia de grupo; el conjunto de grupos resultante constituye el agrupamiento final.

De forma similar a COBWEB, la principal limitación del algoritmo ITERATE es el costo de construcción de la jerarquía de conceptos; en problemas reales con un gran número de objetos, la construcción de dicha jerarquía puede resultar extremadamente costosa. Aunque el algoritmo es capaz de procesar objetos descritos por atributos mezclados, el mejor resultado lo obtiene discretizando los atributos numéricos, por lo que puede perder información importante y afectarse la calidad del agrupamiento obtenido. Por otro la-

do, aunque el proceso de redistribución de los objetos permite mitigar la dependencia del orden de análisis de los objetos, el algoritmo sigue siendo dependiente de este orden.

2.8. Basados en Kmeans

En esta subsección se describen los algoritmos CKM [41], CKMSF [42] y CKMCF [43]. El primero está basado en el algoritmo clásico Kmeans, pero permite trabajar con atributos numéricos y cualitativos, así como generar descripciones de los grupos formados. Los otros dos son extensiones del algoritmo CKM, que mejoran el funcionamiento de este último. Estos tres algoritmos forman un conjunto de grupos disjuntos.

2.8.1. CKM y CKMSF

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos, descritos en términos de un conjunto de atributos cualitativos y/o numéricos $R = \{r_1, r_2, \ldots, r_m\}$. Sea $R_c \subseteq R$ el conjunto de atributos cualitativos que describen a los objetos de O y sea D_j el dominio de valores de cada rasgo $r_j \in R_c$. Para construir el conjunto de grupos y sus descripciones, CKM emplea dos etapas: agregación y caracterización. En la primera etapa se forma la descripción extensional de cada grupo y en la segunda, la descripción intencional o concepto que caracteriza a cada grupo.

La etapa de agregación consta de cuatro pasos. En el primer paso se procesan los objetos de la colección para convertir los atributos cualitativos, que describen a los objetos, en atributos numéricos. La forma común de realizar esta transformación es sustituir, en cada objeto $o_i \in O$, cada rasgo cualitativo $r_j \in R_c$ por un vector de dimension $\left|D_j\right|$, en el cual cada dimension $t-$ésima es un valor Booleano (cero o uno) que significa si el valor del atributo r_j en el objeto o_i era el $t-$ésimo valor de su dominio de valores (D_j) o no. Luego de este primer paso, la descripción de cada objeto $o_i \in O$ es un vector numérico en el cual, cada atributo es un atributo numérico original de o_i o un atributo Booleano resultante de la transformación efectuada en el primer paso. Posteriormente, en el segundo paso se seleccionan aleatoriamente k objetos de la colección y se crean k grupos, cada uno de los cuales contendrá en su descripción extensional uno de estos objetos seleccionados; estos k objetos son además los centroides de cada uno de los grupos creados. Sea $G = \{G_1, G_2, \ldots, G_k\}$ el conjunto de grupos creados y $c(G_i)$ el centroide de cada grupo $G_i \in G$. En el tercer paso se recorre cada objeto $o_i \in O$ de la colección y se calcula la distancia de o_i a cada centroide $c(G_i), i = 1 \ldots k$, asignándose o_i al grupo asociado a su centroide más cercano. La distancia entre dos objetos o_i y o_j se denota por $d^2(o_i, o_j)$ y se calcula como una suma pesada de las distancias que existen entre o_i y o_j, considerando por separado los atributos numéricos originales de cada objeto y los atributos Booleanos.

Las distancias entre dos objetos o_i y o_j, considerando los atributos numéricos originales y los Booleanos, se denotan por $d^2_{\frac{1}{\sigma^2}}(o_i, o_j)$ y $d^2_{\chi^2}(o_i, o_j)$, respectivamente, y se calculan como sigue:

$$d^2_{\frac{1}{\sigma^2}}(o_i, o_j) = \sum_{t=1}^{|R_n|} \frac{\left(o_i(t) - o_j(t)\right)}{\sigma_t^2}, \qquad d^2_{\chi^2}(o_i, o_j) = \sum_{t=1}^{|R_c|} \frac{\left(o_i(t) - o_j(t)\right)}{n_t},$$

donde $|R_n|$ y $|R_c|$ son el número de atributos numéricos originales y Booleanos que incluye cada objeto como parte de su descripción, $o_i(t)$ y $o_j(t)$ son los valores de los objetos o_i y o_j para el atributo t-ésimo, σ_t es la desviación estándar de los valores del atributo t-ésimo y por último, n_j es el número de objetos que tienen a t como valor de su correspondiente atributo.

Cuando todos los objetos son asignado a un grupo, en el cuarto paso se actualiza el valor del centroide de cada grupo $G_i \in G$, considerando los objetos incluidos en G_i. Los pasos tres y cuatro se repiten

iterativamente mientras que el valor de la siguiente función decrezca:

$$W = \sum_{i=1}^{k} \sum_{j=1}^{|O|} p_{o_j}^i \cdot d^2(o_j, c(G_i)),$$

donde $p_{o_j}^i$ es el peso que tiene el objeto o_j dentro del grupo G_i. Si $o_j \in G_i$, entonces $p_{o_j}^i = 1$; en otro caso, $p_{o_j}^i = 0$.

Una vez terminada la etapa de agregación, la etapa de caracterización se encarga de construir los conceptos que describen a cada grupo. Para realizar esto se sigue una estrategia compuesta de dos fases. En la primera fase, se representa cada objeto por los rasgos originales que los describen y se transforman los rasgos numéricos a cualitativos. Para convertir los atributos numéricos a cualitativos, se recorre cada objeto $o_i \in O$, asignándose el valor cualitativo de cada rasgo numérico $r_j \in R$ de o_i de acuerdo a la siguiente función:

$$T(r_j, o_i) = \begin{cases} sup, & \text{si } p_{G_{o_i}}(r_j) + \sigma_{r_j} < o_i(r_j) \\ tip, & \text{si } p_{G_{o_i}}(r_j) - \sigma_{r_j} \leq o_i(r_j) \leq p_{G_{o_i}}(r_j) + \sigma_{r_j} \\ inf, & \text{si } o_i(r_j) < p_{G_{o_i}}(r_j) + \sigma_{r_j} \end{cases},$$

donde $o_i(r_j)$ es el valor del atributo r_j en o_i, G_{o_i} es el grupo al cual pertenece o_i, $p_{G_{o_i}}(r_j)$ es el promedio de valores que tienen los objetos de G_{o_i} para el atributo numérico r_j y σ_{r_j} la desviación estándar del atributo r_j en toda la colección. Una vez codificado cada atributo numérico, se representa cada objeto $o_i \in O$ como un predicado de la forma $(r_1, o_i(r_1)) \wedge (r_2, o_i(r_2)) \ldots \wedge (r_m, o_i(r_m))$ y se realiza un proceso con el objetivo de encontrar la descripción de cada grupo. Para realizar este proceso, CKM asume que cada atributo cualitativo del objeto tiene asociado un retículo de generalización, que es construido por el usuario a partir del conocimiento disponible.

Definición 7 (retículo). *Un retículo es una estructura $L =< E, \leq, \vee, \wedge, *, \emptyset >$ en la cual:*

- *E es un conjunto conjuntos.*
- *$\leq$ es la relación "menos general que", la cual define un orden parcial sobre E y cumple que $\forall e, f \in E, e \leq f \Rightarrow e \subseteq f$.*
- *$*$ es el mayor valor del retículo y $\emptyset$ el menor.*
- *$\forall e, f \in E$ tiene un "mayor" denotado por $e \vee f$ e interpretado como la generalización de e y f; y un "menor" denotado por $e \wedge f$.*

Sea $r_j \in R$ un atributo cualitativo. El retículo de generalización del atributo r_j es un retículo en el cual E es el conjunto de todos los posibles subconjuntos de D_j; siendo D_j el dominio de valores del atributo r_j. En la figura 1 se muestran dos ejemplos de retículo de generalización. En la figura 1(a) se muestra un posible retículo de generalización que tiene el atributo *color-pelo*, asumiendo que su dominio es el conjunto $\{rubio, castaño, negro\}$. En la figura 1(b) se muestra el retículo que propone CKM para los atributos numéricos transformados en cualitativos.

En la segunda fase de la etapa de caracterización, se recorre cada grupo $G_i \in G$ y se genera su descripción. Para encontrar el concepto que describe a G_i se realiza un proceso compuesto de siete pasos. Sea $P_{G_i} = \{p_1, p_2, \ldots, p_l\}$ el conjunto de predicados que representan a los objetos del grupo $G_i \in G$. En el primer paso de este proceso se obtiene el conjunto $P'_{G_i} = \{p_1, p_2, \ldots, p_l\}$ resultante de haber generalizado cada par de predicados del conjunto P_{G_i}; la generalización de un par de predicados, si es que ésta existe, es el predicado que se obtiene generalizando cada uno de los atributos que componen a dichos predicados, utilizando el retículo de generalización de dichos atributos. En el segundo paso se eliminan de P'_{G_i} aquellos predicados que cubren más de α objetos que no están en G_i; α es un parámetro predefinido del algoritmo.

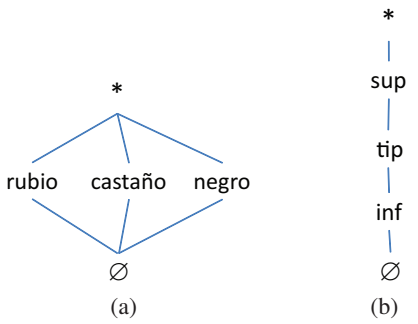

Fig. 1. Ejemplos de retículos de generalización.

Posteriormente, en el tercer paso se adicionan a un conjunto P' todos los predicados de P'_{G_i} y aquellos predicados de P_{G_i} que no pudieron generalizarse o que su generalización fue eliminada de P'_{G_i}. En el cuarto paso, se sustituye el valor de P_{G_i} por el de P'_{G_i} y se repiten otra vez los pasos anteriores, desde el primer paso, mientras que $P_{G_i} \neq \emptyset$. Una vez que $P_{G_i} = \emptyset$, en el quinto paso se eliminan de P' todos los predicados que cubran menos de β objetos de G_i; β es otro parámetro predefinido del algoritmo. Posteriormente, en el sexto paso se elimina de cada predicado los atributos con valor * y se ordenan los predicados resultantes en orden descendente del número de objetos de G_i que cada uno cubre. En el séptimo paso se recorren los predicados ordenados en el paso anterior, seleccionando todo aquel que cubra al menos un objeto no cubierto por los predicados seleccionados hasta ese momento. Sea P'' el conjunto resultante del paso anterior. La descripción de G_i está formada por la disyunción de los predicados de P''.

El algoritmo CKMSF utiliza la misma estructura general que CKM para obtener el conjunto de grupos y sus descripciones, pero incluye dos variaciones en cada etapa. En vez del procedimiento utilizado por CKM en la etapa de agregación, CKMSF utiliza al algoritmo KMSF [44] para construir los grupos en esta etapa. KMSF es una extensión del algoritmo Kmeans clásico, que incluye una función de semejanza que le permite procesar objetos descritos por rasgos mezclados e incompletos, sin necesidad de convertir los atributos cualitativos a numéricos. La otra modificación es en la etapa de caracterización, en la cual CKMSF propone un nuevo retículo de generalización para los atributos numéricos.

La principal limitación de estos algoritmos es que pueden no generar descripciones para algún grupo cuando no existen predicados que cubren a dos o más objetos del mismo. Además, ambos algoritmos pueden construir descripciones muy grandes, las cuales pueden resultar difíciles de interpretar en aplicaciones reales. Adicionalmente, necesitan conocer de antemano el número de grupos a formar, lo cual por lo general es desconocido en problemas reales. Adicionalmente, CKM depende de cinco parámetros mientras que CKMSF depende de tres. Por último, estos algoritmos requieren de conocimiento externo, expresado a través de los retículos de generalización de cada atributo, los cuales no siempre están accesibles en cada aplicación.

2.8.2. CKMCF

Para obtener el conjunto de grupos CKMCF utiliza en la etapa de agregación la misma estrategia que CKMSF. No obstante, para formar las descripciones de los grupos en la etapa de caracterización CKMCF propone utilizar los rasgos complejos. Los rasgos complejos son combinaciones de valores de atributos, que toman un subconjunto de atributos y que son frecuentes en un grupo pero no lo son en el resto de los grupos.

Sea Γ una función que permite calcular la semejanza entre un par de objetos de acuerdo a un subconjunto de los atributos que los describen. Para obtener los rasgos complejos que caracterizan a un grupo G_i se utiliza una estrategia compuesta de tres pasos. En el primer paso se determinan los conjuntos de apoyo relativos a G_i. Los conjuntos de apoyo son subconjuntos de atributos que especifican las partes de los objetos que serán tenidas en cuenta en el calculo de la semejanza entre los mismos. En el algoritmo CKMCF se propone trabajar con:

- Conjuntos Γ-diferenciables: son aquellos en los cuales los objetos de grupos diferentes tienen un valor de semejanza menor si se considera este conjunto que si se considerase el conjunto de atributos completo.
- Conjuntos Γ-caracterizables: son aquellos en los cuales los objetos de un mismo grupo tienen un valor de semejanza mayor si se considera este conjunto que si se considerase el conjunto de atributos completo.
- Conjuntos Γ-testores: son aquellos que son a la vez Γ-diferenciables y Γ-caracterizables.

Para calcular los conjuntos de apoyo, CKMCF propone utilizar un algoritmo genético. En este algoritmo, cada individuo representa un subconjunto de atributos y como función de aptitud se usa el grado en que el subconjunto satisface la propiedad de ser Γ-diferenciable, Γ-caracterizable o Γ-testor. Los individuos de la última generación serán los conjuntos de apoyo. Una vez calculados los conjuntos de apoyo, en el segundo paso se determinan los valores de los atributos expresados por dichos conjuntos de apoyo; la combinación entre los atributos y los valores constituye un rasgo complejo.

Definición 8 (rasgo complejo). *Sean $\Omega = \{r_1, r_2, \ldots, r_p\}$ un conjunto de apoyo y $A = \{a_1, a_2, \ldots, a_p\}$ los valores asociados a los atributos $r_1, r_2, \ldots, r_p$. Se dice que Ω y A forman un rasgo complejo de un grupo G_i ssi se cumple que: i) $\sum_{o_t \in G_i} \Gamma(o_t(\Omega), A) \geq \beta$ y ii) $\sum_{o_t \notin G_i} \Gamma(o_t(\Omega), A) \leq \lambda$; donde $o_t(\Omega)$ es la descripción del objeto o_t considerando solo los atributos de Ω, Γ es una función de semejanza entre dos sub-descripciones de objetos, β es el menor valor de semejanza que deben tener los objetos de un mismo grupo y λ es el mayor valor de semejanza que deben tener objetos de grupos diferentes.*

Una vez determinados los rasgos complejos que describen a G_i, en el tercer paso estos rasgos complejos son representados como predicados y son filtrados, utilizando el sexto y séptimo paso de la etapa de caracterización del algoritmo CKM. El concepto que describe al grupo G_i se obtiene a partir de la disyunción de los predicados obtenidos en el paso anterior.

La principal limitación de este algoritmo es que construye descripciones muy grandes, que pueden resultar difíciles de interpretar en aplicaciones reales. Adicionalmente, de forma similar a CKM y CKMSF, este algoritmo necesita conocer de antemano el número de grupos a formar, lo cual por lo general es desconocido en problemas reales. Por último, es importante mencionar que aunque con el uso de los rasgos complejos CKMCF resuelve los problemas que tiene CKMSF, el uso de estos rasgos impone un mayor tiempo de procesamiento al tener que calcular primero los conjuntos de apoyo y luego los propios rasgos complejos.

2.9. Basados en teoría de grafos

En esta sección se describen los algoritmos LC [45] y RGC [46]. Ambos algoritmos están basados en el Enfoque Lógico-Combinatorio del Reconocimiento de Patrones (LCPR, por sus siglas en inglés) [8] y, a diferencia de otros algoritmos conceptuales, LC y RGC son capaces de procesar objetos descritos por atributos mezclados e incompletos. Adicionalmente, ambos algoritmos construyen la descripción extensional de los grupos utilizando conceptos de teoría de grafos.

2.9.1. LC

Para construir un agrupamiento conceptual a partir de una colección de objetos, LC emplea dos etapas. En la primera se construye, utilizando conceptos de teoría de grafos, la descripción extensional de cada grupo del agrupamiento. Posteriormente, en la segunda etapa se construyen la descripción intencional de cada grupo.

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos, descritos en términos de un conjunto de atributos que pueden ser cualitativos, (i.e., Booleanos, multi-valuados, lingüísticos, etc.), numéricos (i.e., enteros, reales, intervalos) o incluso puede desconocerse su valor. Sea además Γ una función tal que $\Gamma(o_i, o_j)$ establece el grado se semejanza que tiene un par de objetos $o_i, o_j \in O$. El algoritmo LC representa la colección de objeto como un grafo en el cual los objetos de la colección son los vértices del grafo y existe una arista entre dos objetos o_i y o_j si $\Gamma(o_i, o_j) \geq \beta$ o $\Gamma(o_j, o_i) \geq \beta$; donde β es un número real en el intervalo $[0,1]$ que determina el umbral mínimo de semejanza que deben tener dos objetos para ser considerados como semejantes. A partir del grafo anterior, los grupos son aquellos subconjuntos de O que satisfacen un conjunto de propiedades relativas a las semejanzas entre los objetos que constituyen cada subconjunto; i.e., estas propiedades se conocen como *criterio de agrupamiento*. El algoritmo LC propone dos criterios de agrupamiento que forman agrupamientos disjuntos. Uno de estos criterios forma grupos que son *componentes* β-*conexas* y el otro, grupos que son *conjuntos* β-*compactos*.

Definición 9 (componente β-conexa). *Un conjunto $C \subseteq O, C \neq \emptyset$ es una componente β-conexa ssi cumple las siguientes condiciones:*

- $\forall o_i, o_j \in C, \exists o_1, o_2, \ldots, o_q \in C$ *tal que* $o_i = o_1$, $o_j = o_q$ *y* $\forall p \in \{1, 2, \ldots, q-1\}$, *se cumple que* $\Gamma(o_p, o_{p+1}) \geq \beta$.
- $\forall o_i \in O$, *se cumple que si* $o_j \in C$ *y* $\Gamma(o_i, o_j) \geq \beta$, *entonces* $o_i \in C$.
- *todo objeto β-aislado constituye una componente β-conexa. Un elemento $\emptyset_i \in O$ es β-aislado si* $\nexists o_j \in O, o_j \neq o_i$ *tal que* $\Gamma(o_i, o_j) \geq \beta$ *o* $\Gamma(o_j, o_i) \geq \beta$.

Definición 10 (conjunto β-compacto). *Un conjunto $C \subseteq O, C \neq \emptyset$ es un conjunto β-compacto ssi cumple las siguientes condiciones:*

- $\forall o_j \in O$, *si* $o_i \in C$ *y se cumple además que* $(\text{máx}_{o_t \in O, o_t \neq o_i}\{\Gamma(o_i, o_t)\} = \Gamma(o_i, o_j) \geq \beta)$ *o* $(\text{máx}_{o_t \in O, o_t \neq o_i}\{\Gamma(o_j, o_t)\} = \Gamma(o_j, o_i) \geq \beta)$, *entonces se cumple que* $o_j \in C$.
- $\forall o_i, o_j \in C \exists o_1, o_2, \ldots, o_q \in C$ *tal que* $o_i = o_1$, $o_j = o_q$ *y* $\forall p \in \{1, 2, \ldots, q-1\}$, *se cumple que* $\big(\text{máx}_{o_t \in O, o_t \neq o_p}\{\Gamma(o_p, o_t)\} = \Gamma(o_p, o_{p+1}) \geq \beta\big)$ *o* $\big(\text{máx}_{o_t \in O, o_t \neq o_{p+1}}\{\Gamma(o_{p+1}, o_t)\} = \Gamma(o_{p+1}, o_p) \geq \beta\big)$.
- *todo objeto β-aislado constituye un conjunto β-compacto.*

Una vez determinada la descripción extensional de cada grupo, en la segunda etapa LC construye el concepto que describe intencionalmente a cada uno de estos grupos. Para esto, LC utiliza el concepto de l-complejo (ver definición 3) definido por Michalski para la familia de algoritmos CLUSTER, así como el operador Refunion (ver definición 5) y el método GEN, utilizados también por estos algoritmos. En este caso, LC propone describir un grupo $g_i \in G$ por los l-complejos que se forman utilizando los atributos determinados por los testores típicos de G_i, respecto al resto de los grupos.

Definición 11 (testor). *Sea $T = \{r_1, r_2, \ldots, r_s\}$ un conjunto de atributos. T es un testor ssi al considerar solo el conjunto de atributos determinado por T se cumple que $\forall o_i, o_j$ si $\Gamma(o_i, o_j) \geq \beta$ o $\Gamma(o_j, o_i) \geq \beta$, entonces se tiene que o_i y o_j pertenecen al mismo grupo.*

Definición 12 (testor típico). *Un testor T es típico si al eliminar un atributo cualquiera del testor, este pierde la propiedad de ser testor.*

Con el objetivo de buscar conceptos que sean fáciles de interpretar, LC propone buscar los testores típicos de tamaño mínimo presentes en cada grupo; el tamaño de un testor está determinado por el número de atributos que lo forman. Para obtener dichos testores, LC utiliza el algoritmo genético propuesto en [47]. Una vez obtenidos los testores típicos para cada grupo, se forma un l-complejo con cada testor, utilizando los atributos que forman al testor. Cada l-complejo formado es procesado con el operador Refunion y generalizado y simplificado con una variación del método GEN, propuesto por Michalski para CLUSTER/2. El concepto que describe a cada grupo se forma con la disyunción de los l-complejos resultantes del procedimiento anterior.

Aunque la determinación de la descripción extensional de los grupos es computacionalmente poco compleja, la etapa de caracterización que ejecuta LC para formar los conceptos es computacionalmente costosa, principalmente debido al cálculo de los testores. Lo anterior constituye la principal limitación del algoritmo LC. Adicionalmente, LC puede construir conceptos de gran tamaño y por lo tanto, difíciles aún de interpretar por los usuarios. Por otro lado, aunque en [45] se menciona que LC puede utilizar criterios de agrupamiento que forman cubrimientos en vez de particiones, en dicho trabajo solo se muestran criterios que producen particiones y aún más, un criterio que produzca cubrimientos puede traer incongruencias en la parte de caracterización, en donde la teoría empleada asume que los objetos pertenecen solo a un grupo. Otra limitación importante de LC es que puede generar conceptos que no cubran a todos los objetos del grupo. Por último, el criterio de agrupamiento basado en componentes β-conexas que propone LC puede formar grupos con baja semejanza intra-grupo.

2.9.2. RGC

Para construir un conjunto de grupos y sus descripciones, RGC emplea una estrategia compuesta de dos etapas. En la primera etapa se construye la descripción extensional de cada grupo y en la segunda, la descripción intencional de dichos grupos.

Para obtener la descripción extensional de cada grupo, tal y como hiciera el algoritmo LC, RGC utiliza un criterio de agrupamiento $\Pi(O,\Gamma,\beta)$, donde O es la colección de objetos, Γ es la función de semejanza y β es el umbral mínimo de semejanza que se exige para considerar a dos objetos semejantes. Aunque el algoritmo RGC está diseñado para trabajar en la primera etapa con criterios de agrupamiento que formen grupos disjuntos o con traslape, el criterio que se propone en [46] es el basado en componentes β-conexas, que construye grupos disjuntos. Aún más, la segunda etapa desarrollada por RGC asume que los grupos formados en la primera etapa son disjuntos.

En la segunda etapa, de forma similar al algoritmo LC, RGC construye la descripción intencional de cada grupo utilizando el concepto de l-complejo introducido por Michalski. No obstante, RGC propone una nueva forma de determinar si un objeto cubre o no a un l-complejo. Bajo este nuevo criterio, un objeto $o_i \in O$ cubre a un l-complejo l si o_i y l "están relacionados" a través del criterio de agrupamiento Π, utilizado en la primera etapa para determinar los grupos. En este contexto que o_i y l estén relacionados significa que, si se considera solo a los atributos expresados por l, entonces de acuerdo al criterio de agrupamiento empleado o_i y l debieran pertenecer al mismo grupo.

Para obtener los conceptos que describen intencionalmente a cada grupo, RGC utiliza una variación del operador Refunion introducido por Michalski. La nueva variación de este operador, denotado por RUE, permite generar para un grupo $G_i \in G$ y un conjunto de apoyo $t = \{r_1, r_2, \ldots, r_q\}$, un conjunto de l-complejos excluyentes para G_i, que tienen la menor dispersión en términos de T, entre todos los l-complejos que cubren a G_i.

Definición 13 (l-complejo excluyente). *Sea G_i un conjunto de objetos de la colección y l un l-complejo. Se dice que l es excluyente para G_i si al menos un objeto $o_j \in G_i$ cubre a l y no existe ningún otro objeto en $O \setminus G_i$ que cubra a l.*

Es importante mencionar que, a diferencia de LC, RGC garantiza que los l-complejos generados por RUE para un grupo G_i, cubren a todos los objetos de éste. Una vez que se han obtenido los l-complejos excluyentes que cubren a cada grupo $G_i \in G$, éstos son generalizados y simplificados utilizando el método GEN empleado por el algoritmo LC. Aunque al aplicar el método GEN sobre cada variable de un l-complejo l, se exige que el l-complejo resultante siga siendo excluyente, el l-complejo final obtenido por GEN para todas las variables de l puede que no lo sea. Es decir, pueden existir objetos fuera de G_i que sean descritos por algún l-complejo generalizado que describe a G_i. Sea $L_{G_i} = \{l_1, l_2, \ldots, l_s\}$ el conjunto de l-complejos generados para el grupo G_i, luego de ser procesados por el método GEN. Con base en el razonamiento anterior, la descripción intencional de G_i es $(l_1 \vee l_2 \ldots \vee l_s) \wedge \neg(\alpha_1 \vee \alpha_2 \ldots \vee \alpha_q)$, donde $\alpha_1, \alpha_2, \ldots, \alpha_q$ son los l-complejos que describen a cada objeto en $O \setminus G_i$, que es cubierto por algún l-complejo $l_t, t = 1 \ldots s$.

La principal limitación del algoritmo RGC es la complejidad computacional, la cual puede ser exponencial producto del operador RUE. Además, los conceptos que se generan pueden ser muy grandes y por lo tanto difíciles de interpretar en problemas reales. Adicionalmente, en ningún momento se especifica cómo se generan los conjuntos de apoyo necesarios para generar los l-complejos excluyentes. Por último, de forma similar a como sucede en el algoritmo LC, el criterio de agrupamiento basado en componentes β-conexas que propone RGC puede formar grupos con baja semejanza intra-grupo.

2.10. EMO-CC

EMO-CC [48,49] es un algoritmo conceptual, que utiliza técnicas de optimización multi-objetivo y multi-modal, basadas en algoritmos evolutivos, para el descubrimiento de sub-estructuras representativas en una base de datos estructural. Las bases de datos estructurales, a diferencia de las no-estructurales o *planas*, pueden verse como un grafo en el cual los nodos representan atributos o características y las aristas entre nodos corresponden con las relaciones entre dichas características. Por lo general estas bases de datos se representan como un *grafo directo y sin ciclos* [38]. Una sub-estructura consiste en un subgrafo de la base de datos, el cual puede representar a un objeto o concepto embebido en los datos. El agrupamiento que forma EMO-CC es disjunto.

Dado un grafo que representa a la base de datos, EMO-CC construye un conjunto de grupos y sus descripciones empleando dos etapas. En la primera etapa se utiliza el algoritmo genético NSGA-II [50] para obtener un conjunto de sub-estructuras de interés (sub-grafos), que determinan un conjunto de grupos de instancias de la base de datos. Para obtener este conjunto de sub-estructuras, NSGA-II comienza creando una población inicial de tamaño N, denotada por P_0. En esta población, cada cromosoma es representado como un árbol y se genera seleccionando aleatoriamente una observación de la base de datos. A partir de P_0 se construye una población descendencia de tamaño N, denotada Q_0, utilizando los operadores de selección, cruzamiento y mutación.

Para realizar la selección, EMO-CC emplea una variación del clásico método de torneo binario [50], utilizando el *operador de acumulación*. Este operador, denotado por α_n, establece que un individuo i de la población es mejor que otro individuo j (i.e., $i \alpha_n j$) si y solo si el *ranking de no-dominancia* de i es menor que el de j, o si, en caso de tener ambos el mismo ranking de no-dominancia, i tiene mayor *distancia de acumulación* que j. El ranking de no-dominancia especifica el nivel de no-dominancia de un individuo i y éste a su vez, expresa el número de individuos que *dominan* a i. Se dice que un individuo i domina a otro

individuo j, si al evaluar ambos en cada una de las funciones objetivo, i tiene menor o igual valor que j en todas las funciones objetivo y en al menos una i tiene estrictamente menor valor que j. Como funciones objetivo se emplean la complejidad y soporte de la sub-estructura. La complejidad de una sub-estructura está asociada con su tamaño y se calcula como la suma del número de nodos y el número de aristas. Por otra parte, el soporte de una sub-estructura se calcula como el número de observaciones de la base de datos en las que ocurre la sub-estructura. Una sub-estructura ocurre en una observación de la base de datos, si el árbol que define a la sub-estructura es un sub-árbol del árbol que define a la observación. La distancia de acumulación es un estimado del perímetro que forman en el espacio de representación, los vecinos más cercanos del individuo.

Para aplicar los operadores de cruzamiento y mutación, EMO-CC utiliza dos parámetros que establecen la probabilidad que tienen los individuos de cruzarse y mutar. El operador de cruzamiento se realiza al intercambiar aleatoriamente dos sub-árboles de cada par de individuos. Por otra parte, el operador de mutación permite eliminar un nodo hoja, cambiar un nodo por otro o adicionar un nodo hoja, todo de forma aleatoria.

Una vez que P_0 y Q_0 están formados, éstos se unen formando la población $R_0 = P_0 \cup Q_0$, de tamaño $2 \cdot N$. Posteriormente, se divide R_0 en conjuntos F_i, de acuerdo al nivel de no-dominancia que tengan los individuos de R_0. A continuación, se forma la población P_1, utilizando los individuos de los primeros q conjuntos F_i que cumplan que $\sum_{i=1}^{q} |F_i| \leq N$. En caso de que en P_1 haya menos de N elementos, se ordenan los elementos del conjunto F_{q+1} descendentemente de acuerdo al operador α_n y se seleccionan los primeros individuos hasta que $|P_1| = N$. A partir de P_1 se forma la población Q_1, de igual forma a como se creó Q_0, repitiéndose todo el proceso descrito anteriormente, hasta que se alcance un número máximo de iteraciones (parámetro del algoritmo) o hasta cuando no haya cambio en la población.

Sea $P = \{S_1, S_2, \ldots, S_N\}$ el conjunto de sub-estructuras obtenidas por el método NSGA-II. En la segunda etapa se lleva a cabo un proceso de filtrado del conjunto G. Para este propósito, se modifica el criterio de no-dominancia de forma que se tenga en cuenta el traslape entre las observaciones en las que ocurre cada sub-estructura; para medir dicho traslape se utiliza el índice Jaccard [26]. El conjunto final de sub-estructuras de interés está formado por aquellas sub-estructuras que pertenecen el nivel 0 de no-dominancia. Cada una de estas sub-estructuras determina un grupo que tiene como descripción intencional a la propia sub-estructura y como descripción extensional al conjunto de observaciones en las que ocurre la sub-estructura.

La principal limitación de este algoritmo es que requiere ajustar valores de tres parámetros lo cual puede resultar complicado en problemas reales con bases de datos estructuradas. Además, no queda claro en el algoritmo que el agrupamiento final incluya a todas las observaciones de la base de datos, por lo que pueden quedar algunos objetos sin agrupar.

2.11. Default clustering y AGAPE

En esta sección se describen los algoritmos Default clustering (DF, por sus siglas en inglés) [51] y AGAPE [52] los cuales forman un agrupamiento disjunto. DF es un algoritmo propuesto por Velcin y Ganascia para inducir conocimiento a partir de un conjunto de objetos, cuyo espacio de representación pueda estructurarse en forma de retículo (ver definición 7). Por otro lado, AGAPE es una extensión de DF, desarrollada explícitamente para el descubrimiento de tópicos en colecciones de documentos. Ambos algoritmos se basan en el concepto de estereotipo, introducido por el publicista W. Lippman [53] y desarrollado posteriormente por H. Putnam [54]. Según estos autores un estereotipo es un conjunto de características que es compartido por un grupo de personas u objetos y que, a diferencia del concepto clásico de prototipo

introducido por E. Rosch [55], no tiene que coincidir con el comportamiento promedio de los objetos del grupo.

Sea D un espacio de representación que puede estructurarse en forma de retículo. Sea $O = \{o_1, o_2, \ldots, o_n\}$ un conjunto de objetos, tal que $\forall i = 1 \ldots n, \delta(o_i) \in D$; siendo $\delta(o_i)$ la descripción del objeto o_i. Para obtener un agrupamiento, DF propone resolver un problema de optimización en el cual se recorre el espacio D para encontrar el mejor conjunto de estereotipos (conceptos) $S = \{s_1, s_2, \ldots, s_k\}$ que mejor cubre al conjunto de objetos O, respecto a alguna medida de semejanza. Tanto para el algoritmo DF como para AGAPE, cada estereotipo $s_i \in S$ es un elemento del espacio de representación D y determina la descripción intencional de un grupo y los objetos que cubren a s_i constituyen la descripción extensional de dicho grupo.

Para obtener el conjunto de estereotipos S, DF propone utilizar la meta-heurística llamada búsqueda Tabú [56], la cual mejora el algoritmo de búsqueda local. Esta heurística se compone de cuatro pasos. Durante la primera iteración, en el primer paso, se inicializa la lista tabú L y se construye un conjunto de estereotipos S_0. Este conjunto inicial esta formado por un conjunto de objetos de la colección, que son seleccionados aleatoriamente, más al estereotipo simbólico Υ; este estereotipo simbólico representa al conjunto de objetos que no son cubiertos por ningún otro estereotipo. Posteriormente, en el segundo paso se construye el conjunto $V(S_0) = \{S_0^1, S_0^2, \ldots, S_0^n\}$, el cual cada $S_0^j, j = 1 \ldots n$ es un conjunto de estereotipos que es "vecino" del conjunto S_0, en la iteración actual. Los conjuntos de estereotipos que se incluyen en $V(S_i)$, se forman utilizando *movimientos* de baja o de alta influencia sobre el conjunto de estereotipos S_0. Un *movimiento de baja influencia* es enriquecer un estereotipo de S_0, adicionándole o eliminándole algún atributo. Por otra parte, un *movimiento de alta influencia* es eliminar o adicionar un estereotipo al conjunto S_0. Es importante mencionar que los movimientos a realizar no deben pertenecer a la lista tabú L; en otro caso, se correría el riesgo de recorrer zonas del espacio de búsqueda que ya han sido visitadas. Con el objetivo de podar el espacio de búsqueda, en el proceso de construcción de $V(S_i)$ solo se tendrán en cuenta a aquellos estereotipos que satisfacen las siguiente dos restricciones: i) los atributos utilizados en un estereotipo no pueden ser utilizados en otro estereotipo del mismo conjunto S_0^j y ii) todo par de atributos de un estereotipo de un conjunto S_0^j están relacionados, de alguna forma, a través de una sucesión de co-ocurrencias de atributos en el conjunto de objetos que son cubiertos por dicho estereotipo.

En el tercer paso, se evalúan los conjuntos de estereotipos incluidos en $V(S_i)$ y se selecciona como solución S_{i+1} a aquella que alcance el mayor valor de la función de calidad. Para evaluar la calidad de un conjunto de estereotipos S, DF utiliza la siguiente función de calidad:

$$H_O(S) = \sum_{o_i \in O} M_{sim}(\delta(o_i), CS(o_i)),$$

donde M_{sim} es la medida de semejanza entre dos objetos del espacio de representación D y $C_S(o_i)$ es la función que asigna a cada objeto o_i, el estereotipo $s' \in S$ que representa su *cubrimiento relativo*. El cubrimiento relativo de un objeto $o \in O$, denotado por $CS(o_i)$ es el estereotipo $s_i \in S = \{s_1, s_2, \ldots, s_k, \Upsilon\}$ tal que:

a) S_i *incluye por defecto* a $\delta(o_i)$.
b) $M_{sim}(\delta(o_i), s_i) > 0$.
c) $\forall s_k \in S, s_k \neq s_i$ se cumple que $M_{sim}(\delta(o_i), s_k) < M_{sim}(\delta(o_i), s_i)$.

Las condiciones b y c establecen que la semejanza entre o_i y su cubrimiento relativo es mayor que cero y es la mayor que existe entre o_i y cualquier otro estereotipo. Sean d, d' dos objetos en el espacio de representación D. Se dice que d incluye a d' ssi se cumple que todos los objetos cubiertos por d' son cubiertos también por d. Sean d, d' dos objetos en el espacio de representación D, tal que d no incluye a

d'. Se dice que *d incluye por defecto* a d' si existe otro objeto $d'' \in D$, tal que d'' completa a d de alguna forma, tal que d incluye a d'.

Una vez determinada la solución S_{i+1}, en el cuarto paso se incluye en la lista tabú L el movimiento opuesto al realizado para obtener el conjunto de estereotipos S_{i+1} a partir de S_i. En este paso, también se eliminan de L aquellos movimientos que hayan estado en la lista por más de t iteraciones. El proceso anterior se repite desde el segundo paso, por un número máximo de iteraciones $MaxIt$, previamente definido. Sea S el conjunto de estereotipos obtenido al término del proceso anterior. Cada estereotipo $s_i \in S$ define la descripción intencional de un grupo en el agrupamiento, el cual tiene como descripción extensional al conjunto de objetos del cual s_i es su cubrimiento relativo. Cualquier objeto no cubierto se adiciona al grupo que determina el estereotipo Υ. Adicionalmente, si un objeto tiene más de un estereotipo que es su cubrimiento relativo se pueden seguir dos variantes. La primera es asignarlo aleatoriamente al grupo definido por uno de sus cubrimientos relativos y la segunda es asignarlo también al grupo definido por el estereotipo Υ.

El algoritmo DF ha sido aplicado en objetos descritos por pares atributo-valor [51], donde los atributos son variables cualitativas, y también en objetos representados como grafos conceptuales [57]. En tal caso, ambos trabajos solo se diferencian en la función M_{sim}, que han definido para calcular la semejanza entre un par de objetos del problema en cuestión.

AGAPE utiliza la misma estrategia general que DF, pero utiliza operadores diferentes a este último en la formación del conjunto $V(S_i)$. En este caso, los operadores utilizados son *merge*, *split* y una iteración del algoritmo Kmeans, en lo siguiente referida como operador *kmeans*.

Los operadores *merge*, *split* se aplican de forma similar a como sucede en el algoritmo COBWEB: *split* divide un estereotipo en $p > 1$ nuevos estereotipos y *merge* une dos estereotipos existentes para formar un nuevo estereotipo. Por otra parte, el operador *kmeans* realiza dos etapas: re-colocación y actualización. En la primera los objetos son asignados al grupo definido por su estereotipo más semejante, de acuerdo a la función de semejanza. En la segunda etapa, los estereotipos de cada grupo son re-calculados, teniendo en cuenta la descripción de los objetos contenidos en dicho grupo.

Aunque el procedimiento de búsqueda empleado tanto por DF como por AGAPE, impide que los algoritmos caigan en mínimos locales, también hace que éstos sean computacionalmente muy costosos y que además, consuman mucha memoria; esto constituye la principal limitación de estos algoritmos. Por otra parte, ambos requieren del ajuste de tres parámetros los cuales dependen de la colección a procesar y pueden resultar complejos de ajustar en problemas reales. Adicionalmente, estos algoritmos tienden a construir muchos grupos y esto puede dificultar el análisis de los resultados. Por último, estos algoritmos no son capaces de procesar objetos descritos por atributos numéricos, ni incompletos.

2.12. Basados en análisis de conceptos formales

En esta subsección se describen un grupo de algoritmos que están basados en análisis de conceptos formales (FCA, por sus siglas en inglés). Dado un conjunto O de elementos (i.e., objetos, grupos de objetos, etc.), descritos en términos de un conjunto de atributos D, un concepto formal es un par (X,Y) tal que $X \subseteq O, Y \subseteq D$ y se cumple que: i) X es el conjunto de todos los elementos de O que se describen unívocamente por el conjunto de atributos Y y ii) Y es el conjunto de atributos que son comunes a todos los elementos de X. El conjunto X es la descripción extensional del concepto formal y el conjunto Y su descripción intencional.

A continuación se describen los principales algoritmos basados en FCA, que permiten construir un agrupamiento conceptual a partir de una colección de objetos.

2.12.1. MCC

MCC [58] es un algoritmo desarrollado para el agrupamiento conceptual de objetos en movimiento, en video-vigilancia. Un objeto en movimiento (MO, por sus siglas en inglés) se representa como un vector de la forma $MO =< \overrightarrow{v_1}, \overrightarrow{v_2}, \ldots, \overrightarrow{v_m} >$, donde $\forall i = 1 \ldots m, \overrightarrow{v_i}$ es un vector de características que describen al objeto MO en el instante de tiempo i. MCC construye un conjunto de grupos que pueden tener traslape siguiendo una estrategia compuesta de tres etapas.

En la primera etapa, se construye un conjunto de modelos (grupos) que representan a los objetos de la colección, utilizando el algoritmo Expectation-Maximization (EM) [59]. EM es un método iterativo que dado un conjunto de objetos, permite calcular la probabilidad de pertenencia de cada objeto a un conjunto de k modelos o distribuciones, cada modelo representando un grupo. Cada distribución o modelo da la probabilidad de que un objeto tenga un conjunto particular de pares atributo-valor, si se supiera que dicho objeto fuera miembro del grupo representado por esa distribución. Para estimar las probabilidades de pertenencia de los objetos a los modelos, así como los valores de los parámetros que describen a los modelos, EM emplea una estrategia compuesta de dos pasos. Inicialmente, se asumen valores iniciales para los parámetros que describen a cada modelo y usando estos valores, se trata en el primer paso (Expectation) de estimar las probabilidades de pertenencia de los objetos a los modelos. Posteriormente, en el segundo paso (Maximization) se trata de re-estimar los valores de los parámetros que describen a los modelos, con base en los valores de probabilidad de pertenencia a los modelos que fueron calculados en el primer paso. Estos dos pasos se repiten hasta alcanzar la convergencia. Es importante mencionar que aunque EM garantiza convergencia, ésta puede ser a un máximo local. Una vez que termina el algoritmo EM, cada objeto es asignado al grupo con el cual alcanza el máximo de probabilidad.

Sea $G = \{G_1, G_2, \ldots, G_k\}$ el conjunto de modelos o grupos determinados en la primera etapa. En la segunda etapa, se forma un conjunto de conceptos formales a partir del conjunto de modelos y un conjunto F de características que describen a los objetos de dichos modelos. Para esto, en el primer paso de esta segunda etapa se determina el valor de relevancia que tiene cada característica $f \in F$, respecto a cada uno de los modelos. Sea $G_i \in G$ un modelo y $f_j \in F$ una característica. La relevancia de f_j respecto a G_i se denota por $\lambda(G_i, f_j)$ y se calcula como sigue:

$$\lambda(G_i, f_j) = \sum_{t=1}^{m} \left| P_{t,i,F} \cdot \log \left(\frac{P_{t,i,F}}{P_{t,i,F-\{f_j\}}} \right) \right|,$$

donde $P_{t,i,F}$ y $P_{t,i,F-\{f_j\}}$ son las probabilidades de que el modelo i-ésimo describa a su objeto t-ésimo, considerando todos los atributos de F y sin considerar el atributo j-ésimo, respectivamente. Mientras más alto $\lambda(G_i, f_j)$, más significante es f_j para el modelo G_i.

En el segundo paso se buscan todos los pares (A,B), con $A \subseteq G$ y $B \subseteq F$ que formen un concepto formal; siendo $F' \subseteq F$ el conjunto de características para las cuales los grupos de G tienen un valor de significancia mayor o igual que un umbral ε.

Una vez formado el conjunto de conceptos formales basados en modelos, en el tercer paso se realiza un proceso de filtrado en el que iterativamente se unen los pares de conceptos formales $C_1 = (A_1, B_1)$ y $C_2 = (A_2, B_2)$, que cumplan que $ConSim(C_1, C_2) > T_{sim}$ y además $A_2 \subseteq A_1$ o $B_1 \subseteq B_2$ (C_1 es un super-concepto de C_2). Cuando se unen C_1 y C_2 el concepto formal que se obtiene es $C' = (A_1 \cup A_2, B_1 \cup B_2)$. $ConSim(C_1, C_2)$ es la función que determina la semejanza entre los conceptos formales C_1 y C_2:

$$ConSim(C_1, C_2) = \sigma \cdot \left(\frac{|A_1 \cap A_2|}{|A_1 \cup A_2|} \right) + (1 - \sigma) \cdot \left(\frac{|B_1 \cap B_2|}{|B_1 \cup B_2|} \right),$$

donde σ es un parámetro que establece la relevancia de cada parte del concepto formal en la función de semejanza.

Una vez terminado el proceso de filtrado, cada concepto formal (A,B) resultante determina un grupo, que tiene como descripción extensional a todos los objetos que pertenecen a cada modelo $G_i \in A$ y como descripción intencional a las características $f_j \in B$. Adicionalmente, MCC brinda la posibilidad de formar un grafo conceptual en el cual los nodos son los conceptos formales resultantes del proceso de filtrado y las aristas se determinan en dependencia de las relaciones de super-concepto (sub-concepto) que existan entre los nodos; el peso de las aristas se determina de acuerdo a la semejanza entre los nodos, calculada por la función *ConSim*.

Entre las limitaciones principales de MCC está que depende de al menos cuatro parámetros, lo cual puede dificultar su aplicación en problemas reales. Además, MCC no puede procesar objetos descritos por rasgos cualitativos ni incompletos. El conjunto de grupos obtenidos por MCC puede ser grande y tener un traslape elevado. Por último, las descripciones de los grupos determinados por MCC dependen del conjunto F seleccionado en la segunda etapa; sin embargo no se brinda ninguna información de cómo seleccionar este conjunto.

2.12.2. GALOIS

GALOIS [60,61] es un algoritmo incremental que, al igual que el algoritmo MCC [58], permite formar grupos de objetos y sus descripciones utilizando conceptos formales. A partir de los conceptos formales, en lo siguiente referidos solo como conceptos, se puede formar un retículo (ver definición 7), utilizando la relación de orden parcial $\leq$. Bajo esta relación, un concepto (X_1,Y_1) es super-concepto de otro concepto (X_2,Y_2), si se cumple que $X_2 \subseteq X_1$ o, equivalentemente, si $Y_1 \subseteq Y_2$. Los grupos de este retículo pueden tener traslape.

Para construir el retículo, GALOIS procesa los objetos de la colección uno a uno, asumiendo que los conceptos almacenados en el retículo nunca son eliminados. De hecho, al procesar un objeto pueden ocurrir varias situaciones:

1. Los conceptos del retículo no son comparables con el concepto que describe al objeto y por lo tanto, no resultan afectados por la incorporación de dicho objeto.
2. Los conceptos del retículo son iguales o incluyen al concepto que describe al objeto y por lo tanto, solo hay que modificar la descripción extensional de cada uno de estos conceptos del retículo.

Además de los cambios que puede producir el procesamiento de un objeto nuevo sobre los conceptos almacenados en el retículo, dicha adición podría provocar también la creación de nuevos conceptos en el retículo. Esta creación tendría lugar siempre y cuando la intersección de la descripción del objeto a procesar con la descripción intencional de algún concepto, forme un conjunto de atributos que no constituye la descripción intencional de ningún otro concepto del retículo. Considerando lo anterior, en el procesamiento de cada objeto $o \in O$, GALOIS compara, para cada concepto $G = (X,Y)$ del retículo, la descripción intencional de todos los nodos que son super-conceptos de G, con la intersección de la descripción de o con el conjunto Y; en lo siguiente, el conjunto resultante de esta intersección será referido como Z. En esta comparación se consideran los siguientes cuatro casos:

a) Existe algún super-concepto de G cuya descripción intencional es subconjunto de Z.
b) Existe algún super-concepto de G cuya descripción intencional es igual a Z.
c) Existe algún super-concepto de G cuya descripción intencional incluye a Z.
d) La descripción intencional de todos los super-conceptos de G son no comparables con Z.

En los casos a y b, GALOIS no crea nuevos conceptos. Por otro lado, para los casos c y d, GALOIS crea nuevos conceptos que tendrán como descripción intencional al conjunto Z y como descripción extensional al conjunto de objetos que se describen unívocamente por el conjunto de atributos Z. Cada vez

que un nuevo concepto M es adicionado al retículo, GALOIS emplea una estrategia para actualizar los enlaces existentes entre los conceptos del retículo. En el primer paso, se forman los conjuntos S y G, considerando solo aquellos conceptos que existían en el retículo antes de la adición de M. En el conjunto S se insertan todos los conceptos más generales del retículo, que son más específicos que M. Por otro lado, en el conjunto G se insertan todos los conceptos más específicos del retículo, que son más generales que M. Posteriormente, en el segundo paso, GALOIS establece enlaces entre M y cada concepto incluido en los conjuntos S y G, eliminando a la vez cualquier enlace que existía entre un concepto incluido en S y otro incluido en G.

La principal limitación de GALOIS es su complejidad computacional, la cual es elevada. Además, este algoritmo puede construir un gran número de conceptos (grupos), por lo que puede resultar poco útil en problemas reales. Para solucionar este problema se ha trabajado en una extensión de GALOIS, denominada α-GALOIS [62,63], que construye el retículo de conceptos utilizando la misma estrategia que GALOIS y posteriormente, trata de comprimir algunas partes de este retículo. Esta versión también tiene una complejidad computacional elevada y además, necesita que los objetos estén etiquetados o clasificados, lo cual es poco probable de tener en problemas reales.

2.12.3. Algoritmo de Hotho y Stumme

En [64] se propuso un algoritmo conceptual que permite agrupar una colección de documentos, utilizando el algoritmo bisecting-Kmeans [65] y el análisis de conceptos formales. Como resultado del proceso, se obtiene un retículo de conceptos (ver definición 7); los grupos de este retículo pueden tener traslape.

Para formar el retículo de conceptos se emplea una estrategia formada de cinco pasos. En el primer paso se preprocesa cada documento de la colección. Como parte de este preprocesamiento, se eliminan las *stop words* (i.e., artículos, preposiciones, adverbios, entre otras) y se sustituyen los términos restantes por los conceptos de WordNet[1] que se relacionan con dichos términos. A continuación, se adicionan también a la descripción del documento, todos aquellos conceptos de WordNet que se relacionen con los conceptos adicionados anteriormente, a través de las relaciones de hiperonímia o hiponímia. Por último, cada documento es representado como un vector de términos (conceptos de WordNet), donde cada término tiene asociado un peso que denota la importancia de dicho término para describir al documento. Para determinar el peso de cada término se utilizó el esquema de pesado *tf-idf* [66], el cual combina la frecuencia del término dentro del documento (tf), con su frecuencia inversa en la colección (idf). La frecuencia que tiene un término en un documento es el número de veces en que dicho término está en el documento. Por otra parte, la frecuencia inversa de un término (idf) es el número de documentos que contienen al término al menos una vez.

Un vez preprocesados los documentos, en el segundo paso se utiliza iterativamente el algoritmo bisecting-Kmeans para formar una partición de N grupos; donde N es un parámetro del algoritmo. Para esto, se ejecuta el algoritmo bisecting-Kmeans sobre la colección de objetos, hasta que el conjunto de grupos en un nivel sea igual a N. Para calcular la semejanza entre dos documentos se utiliza la medida del coseno [67]. Una vez construida la partición, en el tercer paso se determina el centroide de cada uno de los grupos y se elimina, de cada centroide c_i, aquellos términos cuyo peso sea menor que $\alpha \cdot M_{c_i}$; donde α es un número real en el intervalo $[0,1]$, que es parámetro del algoritmo y M_{c_i} es el peso del término de mayor peso del centroide c_i.

Sea $C = \{C_1, C_2, \ldots, C_N\}$ el conjunto de centroides determinados en el tercer paso, que están descritos por los términos del conjunto $T = \{t_1, t_2, \ldots, t_m\}$. En el cuarto paso, se determinan todos los pares $(A,B), A \subseteq C, B \subseteq T$, que sean conceptos formales. En este contexto, cada concepto formal determina un

[1] http://www.cogsci.princeton.edu/ewn/

grupo que tiene como descripción extensional, a la unión de todos los objetos contenidos en los grupos representados por los centroides incluidos en A y, como descripción intencional, al conjunto de atributos contenidos en B. Por último, en el quinto paso se forma un retículo, utilizando el conjunto de conceptos formales determinados en el cuarto paso y la relación de inclusión "$\leq$", existente entre dichos conceptos formales.

La principal limitación de este algoritmo es su alta complejidad computacional, producto del mismo proceso de formación de los conceptos formales. Otra limitación es que el algoritmo necesita ajustar valores para dos parámetros que dependen de la colección a procesar. Estas dos limitaciones pueden dificultar la aplicación del algoritmo en problemas reales. Por último, este algoritmo no puede procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

2.13. HDCC

HDCC [68,69] es un algoritmo jerárquico conceptual, que extiende los algoritmos jerárquicos aglomerativos clásicos como Single-Link [20], Complete-Link [9] y UPGMA [9]. Esta extensión le permite a HDCC además de construir la jerarquía de grupos disjuntos, construir las descripciones intencionales de los grupos de la jerarquía. A diferencia de los algoritmos aglomerativos clásicos que construyen la jerarquía basados solamente en la distancia entre los grupos, HDCC también tiene en cuenta a todos aquellos grupos que contienen objetos descritos por los conceptos que describen a los grupos más cercanos. Para tener en cuenta esta información, este algoritmo recibe como parámetro dos operadores de generalización Δ y Δ^*. El primero permite generar un concepto que generaliza la descripción de dos objetos. El segundo permite generar un concepto que generaliza los conceptos que describen a un par de grupos. El funcionamiento de HDCC es independiente de la forma en que trabajan estos operadores, por lo que su definición queda a criterio del usuario.

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos. Para formar una jerarquía de grupos y sus conceptos, HDCC emplea una estrategia compuesta de cinco pasos. En el primer paso, cada objeto $o_i \in O$ constituye un grupo, que tiene como descripción extensional a dicho objeto y como descripción intencional a $\Delta(o_i, o_i)$. Sea $G = \{G_1, G_2, \ldots, G_n\}$ el conjunto de grupos construido en el primer paso y que determinan el agrupamiento del nivel base de la jerarquía. En el segundo paso, se calcula la distancia entre todos los pares de grupos existentes y se determina el par de grupos G_i, G_j más cercanos; la función para calcular la distancia entre dos grupos se recibe como parámetro. Sea P_i, P_j los conceptos que describen a los grupos G_i y G_j, respectivamente. Sea $P_{ij} = \Delta^*(P_i, P_j)$ el concepto que generaliza a P_i y P_j. Sea además $S(P)$, el conjunto de objetos de O que son cubiertos por un concepto P. En el tercer paso se adiciona a G el grupo G_{ij} que tiene como descripción extensional la unión de las descripciones extensionales de los grupos G_i y G_j, más el conjunto C. El conjunto C contiene a todos los objetos que están en los grupos, que tienen al menos un objeto que pertenece a $S(P_{ij})$; es decir, que es cubierto por el concepto P_{ij}. La descripción intencional del grupo G_{ij} es el concepto P_{ij}. En el cuarto paso, se crea un nodo en la jerarquía, que contiene al grupo G_{ij} y cuyos hijos son los grupos G_i, G_j y todos los grupos G_z cuyos objetos pertenecen al conjunto C. Posteriormente, en el quinto paso se eliminan del conjunto G los grupos G_i, G_j y todos los grupos G_z cuyos objetos pertenecen al conjunto C. El proceso anterior se repite desde el segundo paso, hasta que el agrupamiento del nivel esté formado por un solo grupo que contiene a todos los objetos de la colección.

Es válido mencionar que, a diferencia de los algoritmos aglomerativos clásicos, HDCC permite unir más de dos grupos en cada iteración, por lo que es capaz de formar jerarquías más pequeñas y fáciles de procesar que las construidas por Single-Link, Complete-Link y UPGMA. No obstante, al estar basado en estos algoritmos, arrastra sus limitaciones, como por ejemplo su complejidad computacional, la cual puede

dificultar la aplicación de HDCC en problemas reales. Además, aunque el algoritmo es flexible respecto a la definición de las funciones Δ y Δ^*, este mismo hecho pudiera aumentar la complejidad computacional de HDCC si dichas funciones son costosas. Por otra parte, HDCC exige que los objetos estén descritos en espacios métricos, lo cual puede impedirle procesar objetos que no puedan representarse en dicho espacio. Por último, HDCC no garantiza que los conceptos generados para cada grupo cubran a todos los objetos contenidos en el mismo.

2.14. GCC

GCC [70] es un algoritmo de agrupamiento que permite formar una jerarquía de conceptos. De forma similar a COBWEB, GCC asume que los objetos están descritos por atributos cualitativos y construye conceptos probabilísticos para describir a los grupos de la jerarquía. Los conceptos de cada nodo se forman con los pares atributo-valor que describen a los objetos almacenados en el sub-árbol definido por el nodo y por las probabilidades de estos pares, calculadas a partir de los objetos observados en dichonodo. No obstante, a diferencia de COBWEB, GCC es estático, por lo que si la colección cambia se necesita re-procesar la colección desde cero para actualizar la jerarquía. Los grupos de la jerarquía formada por GCC son disjuntos.

El funcionamiento de GCC está basado en el concepto de *nivel de generalidad*, con el cual GCC logra establecer un orden entre los conceptos probabilísticos que describen a los nodos. El nivel de generalidad de un concepto probabilístico C_k se denota por $\delta(C_k)$ y está definido de la siguiente forma:

$$\delta(C_k) = \frac{PVN(C_k)}{PVN(C) + PBL(C_k)},$$

donde $PVN(C_k)$ y $PBL(C_k)$ denotan a la predecibilidad y predictividad del concepto C_k, respectivamente. Estas magnitudes se calculan de la siguiente forma:

$$PVN(C_k) = \sum_i \sum_j P(C_k \mid A_i = V_{ij})^2, \qquad PBL(C_k) = \sum_i \sum_j P(A_i = V_{ij} \mid C_k)^2,$$

donde i indexa a los atributos que describen a los objetos, j indexa a los valores del dominio del atributo i-ésimo, $P(C_k \mid A_i = V_{ij})$ es la probabilidad de que un objeto pertenezca al nodo descrito por el concepto C_k, dado que se conoce que el atributo i-ésimo del objeto (A_i) toma el valor j-ésimo de su dominio (V_{ij}); $P(A_i = V_{ij} \mid C_k)$ es la probabilidad de que el atributo i-ésimo del objeto (A_i) tome el valor j-ésimo de su dominio (V_{ij}), dado que se conoce que dicho objeto pertenece al nodo descrito por el concepto C_k. El valor de $\delta(C_k)$ está en el intervalo $[0,1]$ y mientras más cercano a 1(0) esté el valor de $\delta(C_k)$, más general (específico) es el concepto C_k.

El nivel de generalidad de una partición P, formada por K conceptos, se denota por $Gen(P,\alpha)$ y se calcula como sigue:

$$Gen(P,\alpha) = (1-\alpha) \cdot PVN(P) - \alpha \cdot PBL(P),$$

donde α es un umbral mínimo de generalidad. Si la partición P tiene un nivel de generalidad menor que α, entonces el valor de $Gen(P,\alpha)$ es negativo; por otro lado, si $Gen(P,\alpha)$ es positivo, entonces la partición tiene una generalidad mayor al umbral α. Las fórmulas para calcular $PVN(P)$ y $PBL(P)$ son las siguientes:

$$PVN(P) = \frac{\sum_{k=1}^{K} PVN(C_k)}{K}, \qquad PBL(P) = \frac{\sum_{k=1}^{K} PBL(C_k)}{K}.$$

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos descritos por atributos cualitativos y una lista $L = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ de valores reales. La jerarquía que forma GCC a partir de la colección O tiene m niveles

y en la misma, el nivel i-ésimo tiene un nivel de generalidad $\alpha_i \in L$. Para formar esta jerarquía, GCC utiliza una estrategia compuesta de cinco pasos. En el primer paso, se inicializa una variable α_i con el valor $\alpha_1 \in L$ y además se inicializa un conjunto P, con los grupos resultantes de considerar a cada objeto como un grupo; el concepto probabilístico que describe a cada uno de estos grupos se construye con base a los pares atributo-valor que describen a cada objeto. Posteriormente, en el segundo paso se determina el grupo C_{min} que tiene la menor generalidad entre todos los grupos de P. En el tercer paso, se calcula la semejanza entre el grupo C_{min} y el resto de los grupos de P, seleccionándose el grupo C' que tiene la mayor semejanza con C_{min}.

La semejanza entre dos grupos C_1 y C_2, denotada por $Sim(C_1,C_2)$, se calcula de la siguiente forma:

$$Sim(C_1,C_2) = \sum_{i=1}^{I} Sim_A(C_1,C_2,A_i),$$

donde I es el número de atributos que describen a los objetos de la colección y $Sim_A(C_1,C_2,A_i)$ denota a la semejanza que existe entre los grupos C_1 y C_2, en términos del atributo A_i. Esta semejanza se calcula de la siguiente forma:

$$Sim_A(C_1,C_2,A_i) = \sum_{j=1}^{J} min\{P(A_i = V_{ij} \mid C_1), P(A_i = V_{ij} \mid C_2)\}.$$

Una vez que se identifican los grupos C_{min} y C', en el cuarto paso éstos son eliminados de P y en su lugar, se inserta el grupo que se obtiene como resultado de la unión de C_{min} y C'. El concepto que describe a este nuevo grupo se forma con base a los conceptos que describen a C_{min} y C'. Posteriormente, en el quinto paso se evalúa el nivel de generalidad de la partición P, utilizando α_i. Si $Gen(P,\alpha_i)$ es negativo, entonces se repite el proceso anterior, desde el segundo paso. En otro caso, si $Gen(P,\alpha_i)$ es positivo, entonces la partición P define el agrupamiento correspondiente al nivel i-ésimo de la jerarquía. Para formar el resto de los niveles se ejecuta todo el procedimiento anterior, desde el segundo paso, utilizando $\alpha_i = \alpha_{i+1}$. La complejidad computacional de GCC es $O(n^2)$.

Entre las limitaciones del algoritmo GCC está que solo puede procesar objetos descritos por atributos cualitativos. Adicionalmente, como sucede con el algoritmo COBWEB, GCC construye conceptos que pueden ser difíciles de interpretar por el usuario final. Por último, la medida de generalidad que utiliza GCC para construir la jerarquía es muy sensible al ruido, lo cual puede afectar la calidad de la jerarquía formada.

2.15. LINNEO+

LINNEO+ [71,72] es un algoritmo de agrupamiento conceptual que permite procesar incrementalmente una colección de objetos. Este algoritmo asume que los objetos pueden estar descritos por atributos cualitativos y cuantitativos e incluso, pueden tener atributos incompletos. Para el procesamiento de este último tipo de atributos, LINNEO+ puede seguir dos alternativas. La primera es tratar de estimar estos valores a partir de conocimiento que se tenga del dominio del problema, asignando algún valor ficticio: promedio, la moda, etc. La segunda alternativa es inducir estos valores a posteriori, a partir de los grupos formados sin tener en cuenta a los objetos de la colección que tengan atributos incompletos en su descripción..

Para describir a los grupos que forma, LINNEO+ propone utilizar al prototipo del grupo. El prototipo de un grupo es un elemento ficticio que describe el comportamiento promedio de los objetos del grupo. Este objeto está compuesto por pares atributo-valor determinados a partir de los objetos del grupo. Para

los atributos cuantitativos se calcula el promedio y para los cualitativos se utilizan todos los valores, especificando la razón de objetos del grupo que tiene cada uno de los valores.

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos descritos por atributos que pueden ser cualitativos y cuantitativos. Para formar un agrupamiento disjunto LINNEO+ utiliza una estratega compuesta de cinco pasos, que es similar a la que emplea el algoritmo Single-Pass [73]. En el primer paso, se selecciona el primer objeto de la colección y se forma un grupo; el prototipo de este grupo es el propio objeto. En el segundo paso, para cada objeto $o_i \in O$ se calcula la distancia de o_i al prototipo de cada uno de los grupos existentes, seleccionándose el grupo más cercano; sea este grupo G_j. Para calcular la distancia entre un objeto y el prototipo de un grupo, se utiliza la distancia de Hamming generalizada [71]. Esta función de distancia permite tratar a la vez atributos cualitativos y cuantitativos. Es importante mencionar que, con el objetivo de mantener la homogeneidad de las medidas obtenidas con dicha función de distancia, LINNEO+ realiza una normalización en el intervalo $[0, 1]$, de los valores de los atributos cuantitativos.

Sea D_{o_i,G_j} la distancia entre o_i y el prototipo de G_j. Si $D_{o_i,G_j} < \alpha \cdot \varepsilon$, entonces en el tercer paso se adiciona el objeto al grupo G_j y se recalcula el prototipo de G_j, con base en las descripciones de los objetos del grupo; α y ε son dos parámetros predefinidos. Si o_i fue adicionado al grupo G_j, entonces en el cuarto paso se recalcula la distancia entre el prototipo actual de G_j y los objetos del mismo. Como resultado de este paso, todos los objetos de G_j cuya distancia al prototipo actual del grupo sea mayor que α son eliminados del grupo y colocados en una lista L. En otro caso, si $\alpha \cdot \varepsilon < D_{o_i,G_j} < \alpha$, entonces o_i se adiciona en la lista L; este paso permite que las clases se formen inicialmente por objetos bien semejantes y procesar los objetos más lejanos como o_i mas adelante cuando se supone que la clase es más estable. Por último, si $D_{o_i,G_j} \geq \alpha$, entonces se crea un nuevo grupo que contiene a o_i; el prototipo de este grupo es el propio o_i. Cuando todos los objetos de la colección fueron procesados, en el quinto paso se recorre la lista L y se procesan los objetos siguiendo la misma estrategia anterior, específicamente a partir del segundo paso, pero sin modificar el prototipo de un grupo cuando un objeto se adiciona al mismo.

Entre las limitaciones de LINNEO+ está que depende del orden de análisis de los objetos. Aunque la estrategia de re-procesamiento de los objetos de la lista L trata de atacar este problema, el mismo no desaparece pues el algoritmo no especifica qué hacer cuando más de un grupo cumple con ser el más cercano a un objeto. Otra limitación es que LINNEO+ depende de la medida de Hamming generalizada para su funcionamiento, no pudiendo trabajar con otra que se pueda ajustar más al problema real que se está resolviendo. Por último, el algoritmo puede generar conceptos que sean difíciles de interpretar o que dejen sin cubrir a objetos del grupo.

2.16. COING y KIDS

COING [74] y KIDS [75,76] son algoritmos de agrupamiento conceptual que se encargan no solo de construir el mejor subconjunto de todas las posibles clases (grupos) que agrupan a los objetos semejantes, sino de construir todas las posibles clases que contengan a los objetos semejantes; i.e., el *espacio de generalización*. Los grupos o clases de este espacio son disjuntos. Ambos algoritmos asumen que los objetos están descritos por *grafos conceptuales*. Un arco conceptual es una tripleta de la forma: $[\text{concept}_A] \rightarrow (\text{relación}) \rightarrow [\text{concept}_B]$, donde (relación) corresponde a una relación existente entre los conceptos A y B. Un grafo conceptual es un grafo compuesto de arcos conceptuales. En la figura 2 se muestra un dibujo de una casa y una posible descripción de la misma, utilizando un grafo conceptual. Para obtener más información acerca de los grafos conceptuales se puede consultar [77].

Dado un conjunto de descripciones de objetos y un lenguaje de generalización, el espacio de generalización asociado (GS, por sus siglas en inglés) es el conjunto de conceptos *más específicos* que generalizan dichas descripciones (i.e., los objetos). Sea c_1 y c_2 dos conceptos que cubren el mismo conjunto de obje-

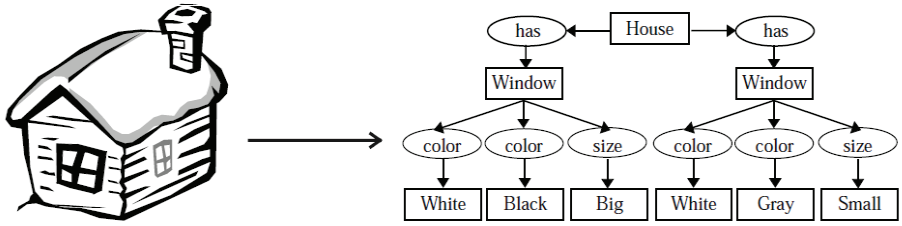

Fig. 2. Ejemplo de una casa y su descripción utilizando un grafo conceptual.

tos. c_1 es mas especifico que c_2 si se cumple que $c_2 \subset c_1$. En el GS, cada nodo o grupo n_i es de la forma $(objs_i, c_i)$, siendo $objs_i$ el conjunto de objetos almacenados en n_i (i.e., descripción extensional del grupo) y c_i el concepto que describe a los objetos de $objs_i$ (i.e., descripción intensional del grupo). El conjunto de conceptos que pertenecen al GS, de conjunto con la relación de subsumisión forman un retículo (ver definición 7).

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos descritos como grafos conceptuales. Para formar el GS, COING utiliza una estrategia compuesta de cuatro pasos. En el primer paso se transforma el grafo conceptual que describe a cada objeto $o_i \in O$, en una lista de arcos conceptuales. Con esta transformación COING se evita el cotejo de grafos el cual es un conocido problema NP-completo, aunque limita el lenguaje de generalización ya que las relaciones existentes entre los arcos no son consideradas. En el segundo paso, auxiliado de retículos de generalización existentes para los diferentes sub-conceptos que forman la descripción de los objetos, COING construye la generalización de los arcos que describen a cada objeto $o_i \in O$. Por ejemplo, los arcos $[ventana] \rightarrow color \rightarrow [negro]$ y $[ventana] \rightarrow color \rightarrow [blanco]$ pueden formar, de acuerdo al retículo de generalización que forman los colores blanco y negro, el arco generalizado $[ventana] \rightarrow color \rightarrow [blanco - negro]$. Posteriormente, en el tercer paso se agrupan los arcos generalizados y los arcos iniciales que cubran el mismo conjunto de objetos de la colección. Este paso permite formar un conjunto de grupos cuya descripción intencional es el conjunto de arcos, generalizados o iniciales, que cubren un mismo conjunto de objetos de la colección y cuya descripción extensional son dichos objetos. Por último, en el cuarto paso se filtra el concepto que describe a cada grupo dejando, de los arcos generalizados, a aquellos que sean más específicos. El conjunto resultante de este último paso representa el agrupamiento conceptual; con dicho conjunto es posible construir un retículo, utilizando la relación de subsumisión.

El algoritmo KIDS utiliza al algoritmo COING como subrutina para construir el GS, utilizando en cada iteración estructuras conceptuales (grafos) cada vez más complejos. Las estructuras conceptuales que utiliza KIDS son *secuencias*, *estrellas* y *holes*. Una secuencia es una sucesión de arcos conceptuales que están unidos unos a otros a través de un concepto común; este concepto común es el destino del primer arco y el origen del segundo. Una estrella es un conjunto de arcos conceptuales que tienen el mismo concepto origen. Un hole es un conjunto de arcos conceptuales que tienen el mismo concepto destino. Una secuencia es de nivel i si esta formada por $i+1$ arcos conceptuales. El concepto anterior se aplica también a las estrellas y a los holes.

Para formar el agrupamiento, KIDS emplea una estrategia compuesta de cinco pasos. En el primer paso se utiliza el algoritmo COING para obtener un GS inicial. En el segundo paso se inicializa una variable $i = 1$ y un conjunto C con todos los grupos del GS determinado en el primer paso. Posteriormente, cada objeto o de cada nodo incluido en el conjunto C es representado en el tercer paso utilizando secuencias, estrellas y holes, de nivel i. Utilizando esta descripción de los objetos, en el cuarto paso se aplica el

algoritmo COING sobre los grupos contenidos en el conjunto C. Como resultado de este paso, pueden adicionarse nuevos grupos al GS o incluso modificarse algunos. Posteriormente, en el quinto paso se vacía el conjunto C y en él se insertan los nodos adicionados al GS en el cuarto paso, así como los que fueron modificados. A continuación, se repite el proceso anterior desde el tercer paso, pero para $i = i+1$. Este proceso se detiene cuando, como resultado del cuarto paso no se adicionan nuevos grupos en el GS, ni se modifican los existentes, o cuando es imposible construir una estructura de nivel $i+1$.

Entre las limitaciones de estos algoritmos está que pueden construir conceptos muy grandes, que son difíciles de interpretar en aplicaciones reales por aquellas personas que no están familiarizadas con la teoría de grafos conceptuales. Otra limitación es que solo pueden procesar objetos descritos por grafos conceptuales. Por último, tanto COING como KIDS tienen una alta complejidad computacional, producto del proceso de construcción del GS, lo que los hacen poco útiles en problemas reales en los cuales el número de objetos es elevado.

2.17. M-DISC

M-DISC [78] es un algoritmo que permite formar una jerarquía de conceptos, a partir de un conjunto de objetos descritos por atributos numéricos. M-DISC emplea una estrategia divisiva, similar a la del algoritmo bisecting-Kmeans [65]. No obstante, a diferencia de este último, M-DISC utiliza una variación de la medida CU definida por el algoritmo COBWEB [23], para guiar el proceso de partición del conjunto de objetos almacenados en cada nodo. La jerarquía formada por este algoritmo esta compuesta de grupos disjuntos.

Para formar la jerarquía de conceptos, M-DISC parte de considerar a todos los objetos como un grupo e iterativamente visita cada nodo y particiona el conjunto de objetos del mismo. Para esto, M-DISC utiliza una estrategia compuesta de tres pasos. En el primer paso, se evalúa si el número de objetos contenidos en el nodo es mayor a un umbral mínimo p_1, dado como parámetro. Si esta condición no se cumple, entonces el nodo no es particionado y el proceso se detiene para este nodo. En otro caso, si el número de objetos contenidos en el nodo es mayor a p_1, entonces se determina la calidad de cada una de las particiones binarias inducidas por cada uno de los atributos que describen a los objetos. Para medir esta calidad, M-DISC utiliza una variación de la medida CU, propuesta en el algoritmo COBWEB.

Sea $O = \{o_1, o_2, \ldots, o_n\}$ un conjunto de objetos y sea $\{C_1, C_2, \ldots, C_k\}$ una partición de k grupos de; conjunto O. La nueva fórmula que utiliza M-DISC para calcular el valor de la medida CU para la partición $\{C_1, C_2, \ldots, C_k\}$ es la siguiente:

$$CU = \frac{RE(O) - \sum_{i=1}^{k} P(C_i) \cdot RE(C_i)}{k},$$

donde $P(C_i)$ es la probabilidad de ocurrencia del grupo C_i, $RE(O)$ y $RE(C_i)$ son los errores de relajación del conjunto de objetos O y del conjunto de objetos almacenados en el grupo C_i, respectivamente. El error de relajación de un conjunto de objetos $O = \{o_1, o_2, \ldots, o_n\}$, descritos por m atributos, se denota como $RE(O)$ y se calcula como sigue:

$$RE(O) = \frac{1}{m} \cdot \left(\sum_{o_i \in O} \sum_{o_j \in O} P(o_i) \cdot P(o_j) \cdot \left[\sum_{t=1}^{m} w_{r_t} \cdot \frac{\left| o_i(r_t) - o_j(r_t) \right|}{\Delta_{r_t}} \right] \right),$$

donde $P(o_i), P(o_j)$ son las probabilidades de ocurrencia de los objetos o_i y p_j, respectivamente; $o_i(r_t), o_j(r_t)$ son los valores que tienen los objetos o_i y p_j para el atributo r_t que los describe, respectivamente; w_{r_t} y Δ_{r_t} son el peso y el factor de normalización del atributo r_t, respectivamente.

Una vez calculada la calidad de cada una de las particiones binarias inducidas, en el segundo paso se selecciona la de mayor calidad y se adicionan los grupos C_1 y C_2 que esta partición forma, como nodos hijos del nodo actual. Sea r_t el atributo que induce la partición de mayor calidad seleccionada en el segundo paso. En el tercer paso, las aristas desde el nodo actual hasta los nodos hijos determinados por C_1 y C_2 se etiquetan con las condiciones que cumplen los objetos de dichos grupos, respecto al atributo r_t. El procedimiento anterior, se repite desde el primer paso, para cada nodo hijo del nodo actual. El concepto que describe a cada nodo de la jerarquía, se forma con las etiquetas que están en el camino desde la raíz hasta el nodo. La complejidad computacional de este algoritmo es $O(m^2 \cdot n^3 \cdot \log n)$.

Entre las limitaciones de M-DISC está que solo trabaja con objetos descritos por rasgos numéricos. Adicionalmente, este algoritmo puede construir conceptos muy largos, que son difíciles de interpretar. Estos conceptos, además pueden tener redundancia ya que el algoritmo no impide que un mismo atributo no se pueda seleccionar más de una vez en el camino desde la raíz a un nodo en específico. El algoritmo es dependiente del orden de análisis de los objetos y su estrategia de agrupamiento puede producir jerarquías muy grandes que pueden resultar difíciles de procesar. Por último, por su complejidad computacional M-DISC puede resultar poco útil en problemas que trabajan con grandes colecciones de objetos.

2.18. LABYRINTH

LABYRINTH [79,80] es un algoritmo de agrupamiento que es incremental y permite formar una jerarquía de conceptos, muy similar a la formada por COBWEB [23]. No obstante, a diferencia de este último, LABYRINTH asume que los objeto son estructurados y se representan por un conjunto de componentes, cada una de las cuales puede ser una *componente primitiva* o una *componente estructurada*. La jerarquía formada por este algoritmo esta compuesta de grupos disjuntos.

Una componente primitiva se describe por un conjunto de pares atributo-valor, donde los atributos son cualitativos. Una componente estructurada se representa en forma de árbol, en el cual la raíz es la componente y tiene tantos nodos hijos como componentes la describan. A su vez estas otras componentes pueden ser primitivas o estructuradas. Adicionalmente, los objetos estructurados contienen también un conjunto de relaciones entre sus componentes.

De forma similar a COBWEB, LABYRINTH utiliza conceptos probabilísticos para describir a los nodos de la jerarquía. Sin embargo, a diferencia de COBWEB, los atributos de los conceptos pueden ser cualitativos o componentes estructuradas. Aquellos conceptos formados solo por componentes estructuradas se conocen por *conceptos estructurados*. Los conceptos estructurados se componen además de un conjunto de relaciones asociadas a las components que forman dicho concepto, cada una de las cuales tiene asociado un trío de valores de probabilidad. El trío de valores asociado a una relación dentro de un concepto estructurado, expresa el número de veces que un objeto clasificado en el nodo, CONFIRMO, NEGÓ o NO-CONTIENE dicha relación.

Para formar la jerarquía de conceptos, LABYRINTH procesa los objetos uno a uno. Cada objeto o_i de la colección es procesado en dos etapas. En la primera etapa, toda componente primitiva que describe a o_i o a alguna otra componente estructurada que describa a o_i, es adicionada a la jerarquía existente, utilizando la misma estrategia de COBWEB. Una vez insertada dicha componente, su valor dentro de la descripción de o_i pasa a ser el nodo en el cual ella fue insertada dentro de la jerarquía. Posteriormente, en la segunda etapa se procesa de forma independiente cada componente estructurada T que describe a o_i, utilizando una estrategia compuesta de dos pasos. En el primer paso se selecciona del árbol que representa a T', toda aquella componente estructurada T' que solo se describa por componentes primitivas. Sea $C = \{T_1^{'}, T_2^{'}, \ldots, T_k^{'}\}$ el conjunto de componentes seleccionadas anteriormente. Es importante observar que, como todas las componentes primitivas que describían a o_i fueron insertadas en la jerarquía y sus

valores sustituidos por el nodo en que fueron insertadas, cada componente $T_i^{'} \in C$ es ahora una componente primitiva. Posteriormente, en el segundo paso cada componente $T_i^{'} \in C$ es insertada en la jerarquía, utilizando una variante de COBWEB, denominada COBWEB' (ver más abajo). Cada vez que una componente $T_i^{'} \in C$ es insertada en al jerarquía, su valor en la descripción de T se sustituye por el nodo en el cual fue insertada. Cuando todas las componentes del conjunto C fueron procesadas, se repite el procedimiento anterior desde el primer paso, hasta que la propia componente T sea una componente primitiva y sea insertada en la jerarquía. Una vez que todas las componentes estructuradas de o_i fueron procesadas, la descripción de o_i puede verse como una componente primitiva y en tal caso, se procesa por COBWEB'.

El funcionamiento de COBWEB' es similar al de COBWEB, lo que ahora se trabaja con componentes estructuradas y esto complica el proceso de inserción del objeto (componente estructurada) dentro de la jerarquía formada. El número de componentes que describen a una componente estructurada T y el número de componentes de un concepto estructurado, que describe a un nodo de la jerarquía, pueden ser diferentes. Por lo tanto, en el proceso de inserción de T en la jerarquía, T puede cotejar de diversas formas con cada concepto almacenado en los nodos de la jerarquía. Para resolver esta parte, COBWEB' incluye una estrategia exhaustiva en la que prueba todas las formas en que pueden cotejar T con el concepto almacenado en un nodo, modificando los valores de probabilidad de las relaciones presentes en el concepto (CONFIRMO, NEGÓ o NO-CONTIENE) según corresponda. Cada vez que COBWEB' prueba una posible forma en que pueden cotejar un concepto y una componente, evalúa el concepto utilizando la siguiente variación de la función CU que utiliza el COBWEB original:

$$CU = \sum_{i=1}^{Atts} \sum_{j=1}^{Values} P(A_i = V_{ij} \mid C_k),$$

donde C_k es el concepto que se está evaluando, $A_i \in Atts$ es un atributo que describe a C_k y V_{ij} es uno de los posibles valores que toma A_i. Luego de probar cada posibilidad, se selecciona aquella que hace que el concepto alcance el mayor valor de calidad. El resto del funcionamiento de COBWEB' es similar al de la versión básica de COBWEB.

LABYRINTH mantiene muchas de las limitaciones presentes en COBWEB. No obstante, su mayor limitación es su muy alta complejidad computacional, debido principalmente al procesamiento recursivo de las componentes estructuradas y por el cálculo que realiza COBWEB' para determinar el mejor entre una estructura y el concepto de cada nodo. Esta alta complejidad dificulta la aplicación de LABYRINTH en problemas reales con grandes volúmenes de objetos.

2.19. Algoritmo de Henry *et al.*

En [81] se propone un algoritmo de agrupamiento conceptual para la identificación de tópicos en colecciones de documentos. Estos tópicos están descritos por dos factores: el conjunto de documentos asociado a cada tópico y la etiqueta que caracteriza a dicho tópico. El primer factor determina la descripción extensional del tópico y el segundo su descripción intencional. Los tópicos formados por este algoritmo son disjuntos. Este algoritmos asume que los documentos de la colección se representan como un vector de términos, en el cual, cada término tiene asociado un peso que denota la importancia de dicho término en la descripción del documento.

Sea $D = \{d_1, d_2, \ldots, d_n\}$ una colección de documentos. Para formar el agrupamiento se utiliza una estrategia compuesta de cuatro pasos. En el primer paso se determina la lista L de pares de términos que co-ocurren en al menos un documento de la colección. Posteriormente, en el segundo paso se calcula probabilidad de generación que tiene cada par de términos $\{t_i, t_j\} \in L$ y se ordena L descendentemente, de

acuerdo a este valor de probabilidad. La probabilidad de generación de un par de términos $\{t_i,t_j\} \in L$ en una colección D, se denota como $P(\{t_i,t_j\} \mid D)$ y se calcula como:

$$(\{t_i,t_j\} \mid D) = \sum_{d_t \in D} P(t_i \mid d_t) \cdot P(t_j \mid d_t) \cdot P(d \mid D),$$

donde $P(d \mid D)$ es la probabilidad de seleccionar al documento d_t dentro de la colección D y se asume igual a $\frac{1}{|D|}$; $P(t_i \mid d_t)$ y $P(t_j \mid d_t)$ son las probabilidades de generar los términos t_i y t_j a partir del documento d_t, respectivamente. La probabilidad de generar un término t en un documento $d \in D$ se calcula como sigue:

$$P(t \mid d) = \frac{TF(t,d)}{\sum_{t' \in d} TF(t',d)},$$

donde $TF(t,d)$ es la frecuencia de ocurrencia del término t en el documento d; $TF(t',d)$ se define análogamente.

Cuando se termina de ordenar la lista L, en el tercer paso se visita cada par de términos $\{t_i,t_j\} \in L$ y se construye el tópico asociado a dicho par. Para esto, primeramente se construye el grafo de β-semejanza G_β que representa al conjunto de soporte del par de términos $\{t_i,t_j\}$, utilizando como función de semejanza la medida del coseno [67]; β es un parámetro del algoritmo. El conjunto de soporte de un par de términos $\{t_i,t_j\} \in L$ está formado por todos los documentos de D en los cuales co-ocurre dicho par de términos.

Sea $S : D \times D \to \Re$ una función simétrica que permite calcular la semejanza entre dos documentos de la colección D y β un valor real en el intervalo $[0,1]$, definido con anterioridad.

Definición 14 (grafo de β-semejanza). *Un grafo de β-semejanza se denota por $G_\beta = \langle V, E_\beta \rangle$ y es el grafo no dirigido en el cual $V = D$ y existe una arista no dirigida $(d_i,d_j) \in E_\beta$ ssi $S(d_i,d_j) \geq \beta$.*

Una vez construido G_β, se determina si el conjunto soporte de $\{t_i,t_j\}$ es homogéneo respecto a su contenido. Para esto se calcula la entropía de la partición formada por las componentes β-conexas de G_β (ver definición 9). Sea $\{C_1,C_2,\ldots,C_q\}$ el conjunto de componentes β-conexas de G_β, la entropía de este conjunto se denota como $H(\{C_1,C_2,\ldots,C_q\})$ y se calcula como sigue:

$$H(\{C_1,C_2,\ldots,C_q\}) = -\sum_{i=1}^{q} P(C_i \mid \{C_1,C_2,\ldots,C_q\}) \cdot \log P(C_i \mid \{C_1,C_2,\ldots,C_q\}),$$

donde $P(C_i \mid \{C_1,C_2,\ldots,C_q\})$ se calcula de la siguiente forma:

$$P(C_i \mid \{C_1,C_2,\ldots,C_q\}) = \frac{|C_i|}{\sum_{j=1}^{q} |C_j|}.$$

Si la entropía de esta partición es menor que 1, entonces el conjunto soporte de $\{t_i,t_j\}$ es homogéneo respecto a su contenido. En tal caso, se construye un tópico que tiene como descripción intencional al par de términos $\{t_i,t_j\}$ y como descripción extensional al conjunto de documentos de D que son relevantes a dicho par de términos. El conjunto de documentos relevantes a un par de términos $\{t_i,t_j\}$ está formado por los documentos contenidos en la componente β-conexa de mayor tamaño de G_β, más todo aquel documento de D cuyos documentos más semejantes pertenezcan a dicha componente. Finalmente, si para el par de términos $\{t_i,t_j\}$ en proceso se construyó un tópico, entonces en el cuarto paso se elimina de D el conjunto de documentos incluidos en la descripción extensional de dicho tópico.

El proceso de generación de tópicos se detiene cuando la colección D esté vacía o cuando se procesaron todos los pares de términos $\{t_i,t_j\}$ incluidos en L. Si al terminar este proceso, todavía quedan documentos

que no pertenecen a ningún tópico creado, entonces para cada uno de estos documentos se crea un tópico que contiene como descripción extensional a dicho documento y como descripción intencional al par de términos más frecuente del mismo.

Existe una versión de este algoritmo, publicada en [82], que incluye algunos cambios con respecto a lo explicado anteriormente. El primer cambio radica en que el umbral β, utilizado para construir el grafo G_β asociado a cada par de términos, ya no es un parámetro sino que se propone una fórmula para calcular su valor. La fórmula propuesta para calcular el valor β es la siguiente:

$$\beta = \frac{1}{|D|} \cdot \sum_{d_i \in D} \frac{1}{k} \cdot \sum_{d' \in V(k,d)} S(d,d'),$$

donde $V(k,d)$ es el conjunto de k documentos más semejantes a d; k es igual a $\lfloor \sqrt{|D|} \rfloor$; $S(d,d')$ es la semejanza que existe entre los documentos d y d'. Otro cambio introducido por esta variante del algoritmo, radica en la forma de calcular la descripción intencional del tópico definido por cada par de términos $\{t_i,t_j\}$.

Sea $R(\{t_i,t_j\})$ el conjunto de documentos relevantes del par de términos $\{t_i,t_j\}$. En esta variante, la descripción intencional del tópico definido por $\{t_i,t_j\}$ es el conjunto de términos cuya correlación con los documentos de $R(\{t_i,t_j\})$ supera un umbral previamente especificado. La última modificación introducida por esta variante del algoritmo está relacionad con cómo se calcula la descripción intencional de los tópicos unitarios creados para cada documento que queda sin agrupar al final del algoritmo. En esta variante, en vez de emplear el par de términos más frecuente del tópico unitario, se utiliza el par de términos de mayor probabilidad de generación.

La principal limitación del método original, así como de la variante propuesta en [82], es que solo son aplicables a documentos, no siendo capaces de procesar objetos descritos por rasgos numéricos, mezclados, ni incompletos. Adicionalmente, la variante propuesta en [82] puede producir descripciones intencionales que pueden ser difíciles de interpretar, debido al gran número de términos que pueden formarla.

2.20. Algoritmo de Aurora *et al.*

En [83] se propuso un algoritmo jerárquico incremental para el procesamiento de noticias. Este algoritmo se encarga de construir tanto los grupos de noticias como las descripciones de dichos grupos. En este contexto, se asume que cada noticia se describe por tres vectores de términos: uno que describe la noticia por las palabras que contiene y los otros dos, por los lugares y fechas que se mencionan en la noticia, respectivamente. Cada uno de los términos incluidos en estos vectores tiene asociado un valor numérico o peso que determina la importancia de dicho término para describir a la noticia. Los grupos formados por este algoritmo son disjuntos.

Sea $D = \{d_1, d_2, \ldots, d_n\}$ una colección de noticias. Para construir la jerarquía de grupos, el algoritmo emplea un enfoque aglomerativo. Bajo este enfoque, cada noticia de la colección constituye un grupo del nivel base de la jerarquía. A partir de este punto, se calculan los centroides de cada uno de los grupos del nivel base y se obtiene el agrupamiento del siguiente nivel aplicando un algoritmo de agrupamiento sobre el conjunto de centroides. Este proceso se repite iterativamente hasta que se alcanza la condición de parada; en ese punto, la jerarquía queda formada por los niveles construidos hasta ese momento. El algoritmo que se propone utilizar para obtener el agrupamiento de cada nivel es el algoritmo Compacto Incremental (CI) [84]. Este algoritmo representa el conjunto de centroides a agrupar como un grafo de máxima β-semejanza y determina el conjunto de conjuntos compactos de dicho grafo (ver definición 10);

en este contexto, cada conjunto compacto determina un grupo del agrupamiento. β es un parámetro del algoritmo.

Definición 15 (grafo de máxima β-semejanza). *Un grafo de máxima β-semejanza, denotado por* $G_{max-\beta} = \langle V, E_{max-\beta} \rangle$*, es el grafo dirigido en el cual* $V = D$*,* $E_{max-\beta}$ *es el conjunto de aristas y este cumple que* $\langle d_i, d_j \rangle \in E_{max-\beta}$ *ssi* $S(d_i, d_j) = max\{S(d_i, d_k) \mid d_k \in V \wedge d_k \neq d_i \wedge S(d_i, d_k) \geq \beta\}$.

Para construir $G_{max-\beta}$ se utiliza una función de semejanza que utiliza la información de cada uno de los vectores que describe a la noticia. La condición de parada del proceso de construcción de la jerarquía es que una vez construido $G_{max-\beta}$, se cumpla que $E_{max-\beta} = \emptyset$.

Dado que el algoritmo propuesto es incremental, es capaz de procesar adiciones de noticias a la colección. Cada vez que se adiciona una noticia nueva a la colección, esta forma un nuevo grupo en el nivel base de la jerarquía y representa un nuevo centroide a procesar en el nivel siguiente. Este nuevo centroide puede modificar la estructura del grafo que representa a las noticias del nivel siguiente, por lo que se ejecuta el algoritmo CI para actualizar dicho agrupamiento. Esta actualización puede incluir la creación de grupos nuevos, así como la eliminación o modificación de grupos existentes. Cada uno de estas acciones conlleva a cambios en el nivel siguiente. Una adición significa adicionar un centroide al grafo del siguiente nivel. Una eliminación representa un centroide que hay que eliminar del grafo del nivel siguiente. Por último, una modificación representa un centroide que hay que eliminar del grafo del siguiente nivel y luego volver a adicionarlo, considerando los nuevos cambios en su descripción. Para actualizar el agrupamiento del siguiente nivel se utiliza también el algoritmo CI. Este procedimiento de actualización continúa hasta alcanzar la condición de parada.

Una vez terminada la construcción de la jerarquía, se recorre cada nivel de ésta y se construye el concepto o etiqueta que describe a cada uno de los grupos del nivel. Para realizar esto, en cada grupo se seleccionan los términos más frecuentes del grupo, de acuerdo al peso asociado a cada término, que no son frecuentes en el resto de los grupos del nivel. Posteriormente, utilizando estos términos frecuentes se determinan los testores típicos de cada grupo (ver definición 12). A continuación, se extraen de las noticias contenidas en el grupo, aquellas oraciones en las que estén presentes los términos de los testores típicos. El concepto que describe a cada grupo se compone de las oraciones que contengan más términos de los testores típicos, dentro del conjunto de oraciones extraídas en dicho grupo.

Entre las limitaciones de este algoritmo es que construye jerarquías compuestas de muchos niveles y muchos grupos, que pueden resultar difíciles de procesar en problemas reales. Por otro lado, los conceptos que se generan para cada grupo son muy grandes y por lo tanto, resultan difíciles de interpretar. Adicionalmente, aunque la complejidad computacional del proceso de construcción de la jerarquía es $O(n^3)$, lo cual no es ciertamente bajo, la complejidad del proceso de extracción de los conceptos de los grupos puede ser exponencial y por lo tanto, esto dificulta la aplicación del algoritmo en problemas con un gran número de noticias. Por último, un aspecto que puede limitar la aplicación de este algoritmos en problemas reales es que solo puede procesar noticias, o documentos, en el caso más general.

2.21. Basados en patrones frecuentes

En esta subsección se describen un conjunto de algoritmos que basan su funcionamiento o gran parte de este, en el descubrimiento de patrones frecuentes en una colección de objetos.

De forma general, un patrón es una estructura que forma parte de la descripción de un objeto. Estas estructuras pueden ser conjuntos, secuencias, árboles o grafos, entre otras. Luego, un patrón será frecuente en un conjunto de objetos, si y solo si el número de objetos en los que está presente dicho patrón supera un umbral previamente establecido. El número de objetos en los que está presente un patrón se conoce por

soporte. Es importante mencionar que el soporte de un patrón puede también interpretarse como la razón entre el número de objetos en los que está presente dicho patrón y el número total de objetos del conjunto de estudio.

A continuación se describen los principales algoritmo basados en patrones frecuentes, que permiten construir un agrupamiento conceptual, a partir de una colección de objetos.

2.21.1. FTC, HFTC, FTSC y FTSHC

FTC [85] y HFTC [85] son algoritmos diseñados para el procesamiento de colecciones de documentos. FTC construye un conjunto de grupos mientras que HFTC construye una jerarquía de grupos, utilizando al algoritmo FTC. Los grupos que forma cada algoritmo tienen una descripción extensional y una intencional. Ambos algoritmos asumen que cada documento está representado como un vector de términos, donde cada término tiene asociado un peso que determina la importancia del término para describir al documento; en estos algoritmos, el peso de cada término se calcula como la frecuencia absoluta del término en el documento; i.e., el número de veces en que está el término en el documento. Tanto FTC como HFTC están basados en el concepto de conjunto de términos frecuentes.

Sea $D = \{d_1, d_2, \ldots, d_n\}$ una colección de documentos, descritos por un conjunto de términos $T = \{t_1, t_2, \ldots, t_m\}$. En lo siguiente, cualquier conjunto de términos $f \subseteq T$ será llamado *itemset*. Un documento $d_i \in D$ contiene a un itemset f si contiene a cada uno de los términos de f. En lo siguiente se denotará como $cov(f)$ al conjunto de documentos $d_i \in D$ que contienen a f.

Definición 16 (soporte de un itemset). *El soporte de un itemset $f \subseteq T$ se denota como $s(f)$ y es el número de documentos $d_i \in D$ que contienen a f; i.e., $s(f) = |cov(f)|$.*

Definición 17 (longitud de un itemset). *La longitud o tamaño de un itemset $f \subseteq T$ se denota por $t(f)$ y es el número de términos que el mismo contiene.*

Definición 18 (conjunto de términos frecuentes). *Sea $f \subseteq T$ un itemset. Se dice que f es frecuente en la colección D si su soporte es mayor o igual a un umbral s_{min}, previamente definido; i.e., $s(f) \geq s_{min}$.*

Para construir un conjunto de grupos y sus descripciones, FTC emplea una estrategia compuesta de cuatro pasos. En el primer paso se construye el conjunto F de itemset frecuentes para un umbral s_{min}, utilizando el algoritmo Apriori [86]. En el segundo paso se determina, para cada itemset $f_i \in F$, el traslape que tiene el conjunto $cov(f_i)$, respecto al conjunto de documentos que se forma por la siguiente operación $\bigcup_{f' \in (F \setminus f_i)} cov(f')$. Para calcular este traslape FTC propone cuatro alternativas [85,87]: el traslape estándar (SO, por sus siglas en inglés), el traslape estándar mejorado (ISO, por sus siglas en inglés), el traslape de entropía (EO, por sus siglas en inglés) y el traslape de entropía mejorado (IEO, por sus siglas en inglés).

El traslape estándar y el traslape estándar mejorado del conjunto $cov(f_i)$, denotados como $SO(cov(f_i))$ y $ISO(cov(f_i))$ respectivamente, se calculan de la siguiente forma:

$$SO(cov(f_i)) = \frac{\sum_{d_j \in cov(f_i)} frec(d_j)}{|cov(f_i)|}, \qquad ISO(cov(f_i)) = \sum_{d_j \in cov(f_i)} (frec(d_j) - k),$$

donde $frec(d_j)$ es el conjunto de itemset frecuentes que están incluidos en el documento d_j; $k = 2^n - 1$, siendo n el número de términos del itemset f_i.

El traslape de entropía y el traslape de entropía mejorado del conjunto $cov(f_i)$, denotados como $EO(cov(f_i))$ y $IEO(cov(f_i))$ respectivamente, se calculan de la siguiente forma:

$$EO(cov(f_i)) = \sum_{d_j \in cov(f_i)} -\frac{1}{frec(d_j)} \cdot \ln\left(\frac{1}{frec(d_j)}\right), \quad IEO(cov(f_i)) = \sum_{d_j \in cov(f_i)} -\frac{1}{m \cdot frec(d_j)} \cdot \ln\left(\frac{1}{m \cdot frec(d_j)}\right),$$

donde $m = \sum_{d_j \in cov(f_i)} \frac{1}{frec(d_j)}$.

En el tercer paso se determina el itemset f_i cuyo conjunto $cov(f_i)$ tiene el menor traslape y se adiciona a un conjunto G, el grupo que tiene como descripción intencional a f_i y como descripción extensional a los documentos en $cov(f_i)$. Posteriormente, en el cuarto paso se elimina el itemset f_i del conjunto F y el conjunto de documentos $cov(f_i)$ de la colección D. El proceso anterior se repite desde el segundo paso, hasta que la colección D quede vacía. El agrupamiento final está formado por cada uno de los grupos insertados en el conjunto G. El conjunto de grupos que forma FTC es disjunto, pero puede modificarse sencillamente para obtener grupos con traslape, si no se tiene en cuenta el cuarto paso anteriormente descrito.

Para construir una jerarquía de grupos, HFTC emplea una estrategia compuesta de tres pasos, en la cual utiliza al algoritmo FTC. En el primer paso se construye el conjunto F de itemset frecuentes para un umbral s_{min}, utilizando el algoritmo Apriori. En el segundo paso, se recorre cada itemset $f \in F$ tal que $t(f) = 1$ y se forma con él un grupo que tiene, como descripción intencional a f y como descripción extensional a $cov(f)$. Posteriormente, en el tercer paso se visita cada grupo G de este nivel y se particiona el conjunto de objetos que están en la descripción extensional de G. Sea f el itemset que constituye la descripción intencional del grupo G. Para particionar el conjunto de objeto en la descripción extensional de G se utiliza el algoritmo FTC y se considera como conjunto de itemset frecuentes al conjunto F' que se forma con cada itemset $f' \in F$ tal que $f \subset f'$ y $t(f') - t(f) = 1$. Este proceso continúa recursivamente por cada nodo de la jerarquía, hasta que no hayan más itemset frecuentes para seguir dividiendo los documentos de un grupo. Es importante mencionar que en la variante de FTC que aplica el algoritmo HFTC para particionar el conjunto de documentos contenido en la descripción extensional de cada grupo, no se tiene el cuarto paso de FTC. Esto es, luego de formar cada grupo, los documentos contenidos en éste no se eliminan de la colección. El efecto de este cambio radica en que ahora HFTC puede formar grupos con traslape.

La principal limitación de FTC y HFTC es la dependencia del parámetro s_{min}. Valores muy bajos hacen al algoritmo muy costoso, mientras que valores muy altos pueden formar agrupamientos de baja calidad, al no descubrir itemsets interesantes. Otra limitación es que la calidad de los conceptos que describen a los grupos puede ser baja, ya que no tienen en cuenta la frecuencia de los términos dentro de los documentos del grupo sino en toda la colección. Por último, estos algoritmos no son capaces de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

FTSC y FTSHC son dos algoritmos reportados en [88,89] que siguen estrategias similares a las del algoritmo FTC. El primero es una modificación del algoritmo FTC, en la cual: i) solo se considera el traslape estándar y traslape de entropía para calcular el traslape del conjunto $cov(f_i)$ y ii) en la estrategia utilizada para formar el agrupamiento se consideran todos los pasos utilizados por FTC, exceptuando el cuarto paso. Por otro lado FTSHC, es un algoritmo que construye una jerarquía a partir del agrupamiento obtenido por FTSC. Sea $G_\beta = \{G_1, G_2, \ldots, G_k\}$ el conjunto de grupos determinado por FTSC. Para formar la jerarquía, se considera como nivel tope de la misma el conjunto de grupos $G' \in G$ que estén descritos por un itemset de tamaño 1. A partir de este punto, los niveles siguientes se forman a partir del nivel anterior. Para formar el nivel $i > 1$, se recorre cada grupo G' del nivel i y se inserta en su lista de nodos hijos todo grupo $G'' \in G$ tal que $f_{G'} \subset f_{G''}$; siendo $f_{G'}$ y $f_{G''}$ los itemset frecuentes que describen intencionalmente a los grupos G' y G'', respectivamente. Esta estrategia continúa hasta que ya no se generan más niveles. Los grupos formados por estos dos algoritmos pueden tener traslape. FTSC y FTSHC tienen las mismas limitaciones que los algoritmos FTC y HFTC, respectivamente.

2.21.2. FIHC, FCDC y FCIHC

FIHC [90,91] es un algoritmo de agrupamiento que está basado en el concepto de conjunto de términos frecuentes y que construye una jerarquía de grupos disjuntos, a partir de una colección de documentos. Cada grupo de esta jerarquía tiene una descripción extensional y otra intencional. FIHC asume que cada documento está representado como un vector de términos, donde cada término tiene asociado un peso que determina la importancia del término para describir al documento; en estos algoritmos, el peso de cada término se calcula con el esquema de pesado *tf-idf* [66].

Para formar el conjunto inicial de grupos, en la primera etapa se emplean dos pasos. En el primer paso se construye el conjunto F de itemset frecuentes para un umbral s_{min}, utilizando el algoritmo Apriori [86]. Posteriormente, en el segundo paso, se recorre cada itemset $f \in F$ y se forma con él un grupo que tiene, como descripción intencional a f y como descripción extensional a $cov(f)$.

Sea d_i un documento y $f_j \in F$ un itemset frecuente. El puntaje de un documento d_i respecto al grupo formado por un itemset f_j se calcula de la siguiente forma:

$$Score(cov(f_j) \leftarrow d_i) = \left(\sum_x n(x) \cdot cluster_{support}(x)\right) - \left(\sum_{x'} n(x') \cdot global_{support}(x')\right),$$

donde x denota a los términos (itemset de tamaño 1) de d_i, que pertenecen a F y que también son frecuentes dentro del conjunto de documentos $cov(f_j)$; x' denota a los términos de d_i, que pertenecen a F y que no son frecuentes dentro del conjunto de documentos $cov(f_j)$; $n(x)$ y $n(x')$ denotan los pesos que tienen los términos x y x' dentro del documento d_i; $cluster_{support}(x)$ denota el soporte que tiene el término x en el conjunto de documentos $cov(f_j)$ y $global_{support}(x')$ denota el soporte que tiene el término x' en toda la colección de documentos.

Dado que un mismo documento puede contener más de un itemset frecuente, el conjunto inicial de grupos puede tener traslape entre los grupos. Con base en esto, en la segunda etapa se realiza un proceso para volver disjunto el conjunto de grupos iniciales. Para lograr esto, se emplean dos pasos. En el primer paso se recorre cada documento $d_j \in D$ y se determina el puntaje que tiene dicho documento respecto a cada uno de los grupos que lo contienen en el agrupamiento inicial. Posteriormente, en el segundo paso cada documento es asignado al grupo con el cual alcanza el mayor puntaje. Si hubiera más de un grupo con el cual un documento alcanza el mayor puntaje, entonces se selecciona aquel cuya descripción intencional sea mayor. Si todavía persiste el empate, entonces se selecciona uno de los grupos aleatoriamente.

Sea $G = \{G_1, G_2, \ldots, G_n\}$ el conjunto de grupos resultantes de la segunda etapa. Para construir la jerarquía en la tercera etapa se sigue una estrategia compuesta de dos pasos. Sea k el tamaño del itemset de mayor tamaño del conjunto F. La jerarquía se construye de abajo hacia arriba, considerando primero los grupos descritos intencionalmente por los itemsets de mayor tamaño. En el primer paso, se forma el nivel base de la jerarquía (nivel k) con los grupos cuya descripción intencional tenga tamaño k. En el segundo paso, para encontrar el nivel inmediato superior (nivel $k-1$), se recorre cada grupo G_i del nivel k y se identifica cuál grupo $G' \in G$ es el mejor "padre" del grupo G_i. El conjunto de todos los grupos que sean los mejores "padres" de los grupos del nivel k, constituye el agrupamiento del nivel $k-1$. Sea G_i un grupo del nivel k de la jerarquía, que tiene como descripción intencional al itemset f_{G_i}. Sea d_{G_i} el documento ficticio que se forma sumando los vectores de todos los documentos almacenados en los grupos que forman el árbol definido por el grupo G_i en la jerarquía. Sea $P = \{G_1, G_2, \ldots, G_q\}$ el conjunto de grupos tal que $P \subseteq G$ y se cumple que cada grupo $G_j \in P$ se describe intencionalmente por un itemset f_{G_j} tal que $f_{G_j} \subseteq f_{G_i}$ y $t(f_{G_i}) = t(f_{G_j}) + 1$. El mejor "padre" del grupo G_i es el grupo del conjunto P para el cual el documento ficticio d_{G_i} alcanza el mayor puntaje.

El procedimiento anterior se repite recursivamente para cada uno de los niveles superiores, hasta alcanzar el nivel 1; este nivel está formado por los grupos descritos por los itemset de tamaño 1. Cada uno de

los grupos del primer nivel es hijo de un nodo ficticio que se inserta como raíz de la jerarquía. Es importante mencionar que el primer nivel de la jerarquía también contiene un grupo que tiene como descripción intencional el término "misceláneas" y en cuya descripción extensional está todo aquel documento de la colección D que no esté incluido en ningún otro grupo de la jerarquía.

Cuando se termina de construir la jerarquía, en la etapa cuatro se realiza un proceso de filtrado que tiene como objetivo reducir el ancho y profundidad de la jerarquía. Para lograr este propósito se aplican dos podas sobre la jerarquía.

Sean G_1 y G_2 dos grupos (nodos) de la jerarquía formada. La semejanza inter-grupo entre G_1 y G_2 se denota como $S_{it}(G_1 \leftrightarrow G_2)$ y se calcula como sigue:

$$S_{it}(G_1 \leftrightarrow G_2) = \sqrt{S(G_1 \leftarrow G_2) \cdot S(G_2 \leftarrow G_1)},$$

donde $S(G_1 \leftarrow G_2)$ y $S(G_2 \leftarrow G_1)$ son las semejanzas del grupo G_2 respecto al grupo G_1 y la semejanza del grupo G_1 respecto al grupo G_2, respectivamente. La semejanza de un grupo G respecto a otro grupo G' se calcula de la siguiente forma:

$$S(G' \leftarrow G) = \frac{Score(G' \leftarrow d_G)}{\sum_x n(x) + \sum_{x'} n(x')} + 1,$$

donde d_G el documento ficticio que se forma sumando los vectores de todos los documentos almacenados en los grupos que forman el árbol definido por el grupo G en la jerarquía; x denota a los términos (itemset de tamaño 1) de d_G, que pertenecen a F y que también son frecuentes dentro del conjunto de documentos que determina la descripción extensional de G; x' denota a los términos de d_G, que pertenecen a F y que no son frecuentes dentro del conjunto de documentos que determinan la descripción extensional de G.

La primera poda que se aplica en el proceso de filtrado tiene como objetivo eliminar de la jerarquía a todo nodo que tenga una alta semejanza con su nodo padre. Para realizar esta poda se recorre la jerarquía de abajo hacia arriba y para cada nodo N, se calcula la semejanza inter-grupo que existe entre él y sus nodos hijos. Si la semejanza inter-grupo que existe entre N y algún nodo hijo T es mayor que 1, entonces se elimina el nodo T de la jerarquía y sus nodos hijos pasan a ser hijos del nodo N.

La segunda poda tiene como objetivo reducir el número de nodos del primer nivel. Para esto, se calcula la semejanza inter-grupo de todos los pares de nodos del primer nivel y se selecciona al par de grupos G_1 y G_2 que alcance la mayor semejanza. Si esta semejanza es mayor que 1, entonces se unen los grupos G_1 y G_2, formando el grupo G'; los nodos hijos de G_1 y G_2 serán ahora los hijos del nodo G'. Cuando se unen dos grupos G_1 y G_2, el grupo resultante tiene como descripción extensional e intencional, a la unión de las descripciones extensionales e intencionales de G_1 y G_2, respectivamente. El procedimiento anterior se repite hasta que el número de grupos del primer nivel sea igual a un parámetro definido previamente.

De forma similar a FTC y HFTC, la principal limitación de FIHC es su dependencia del valor s_{min} para el cálculo de los itemset frecuentes. Adicionalmente, la estrategia de construcción y filtrado de la jerarquía utilizada por FIHC es muy costosa y consume mucha memoria, lo cual dificulta su aplicación en problemas reales. Por último, este algoritmo no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

En [92] se presentó una variante del algoritmo FIHC, llamada FCDC, cuya principal diferencia con FIHC es que ésta incluye una etapa de pre-procesamiento de los documentos. En esta etapa FCDC utiliza WordNet para buscar el concepto asociado a cada término que describe a los documentos. Como resultado, cada documento se representa como un vector de conceptos de WordNet, donde cada concepto tiene asociado un peso que expresaba su importancia en la descripción del documento. El resto de los pasos de FCDC son los mismos que FIHC. Adicionalmente, en [93] se presentó otra variante de FIHC, la cual sigue la misma estructura general de FIHC, pero en vez de utilizar los itemset frecuentes para formar

los grupos, esta variante utiliza los itemset cerrados (ver definición 19). Es válido mencionar que dado que el conjunto de itemset cerrados es menor que el de itemset frecuentes, esta variante logra ciertas aceleraciones respecto a FIHC, en la construcción de la jerarquía. No obstante, tanto esta variante como FCDC, presentan la mismas limitaciones que el algoritmo original FIHC.

2.21.3. TDC

TDC [94] es un algoritmo de agrupamiento que, dada una colección de documentos, permite formar una jerarquía de grupos con traslape, cada uno compuesto por una descripción extensional y otra intencional. Este algoritmo asume que cada documento está representado como un vector de términos, donde cada término tiene asociado un peso que determina la importancia del término para describir al documento; TDC define el peso de cada término utilizando el esquema de pesado *tf-idf* [66]. A diferencia de los algoritmos FTC y FIHC, TDC basa su funcionamiento en el cálculo de los *conjunto de términos cerrados.*

Definición 19 (conjunto de términos cerrados). *Sea $f \subseteq T$ un itemset frecuente. Se dice que f es cerrado si no existe ningún otro itemset frecuente f' tal que $f \subset f'$ y $s(f) = s(f')$.*

Sea $D = \{d_1, d_2, \ldots, d_n\}$ una colección de documentos descritos por un conjunto de términos $T = \{t_1, t_2, \ldots, t_m\}$. Para formar la jerarquía, TDC emplea una estrategia compuesta de cinco etapas. En la primera etapa, se calcula el conjunto FC de itemsets cerrados, utilizando el algoritmo CLOSET+ [95]. Para determinar el umbral s_{min} que se utiliza para calcular a los itemset cerrados, TDC propone una estrategia compuesta de dos pasos. En el primer paso, se ordenan los términos que describen a la colección de documentos, en orden ascendente de acuerdo al soporte de cada término. Sea L la lista ordenada de dichos términos. Posteriormente, en el segundo paso se selecciona el primer término $t \in L$ y se elimina de L y de cada uno de los documentos que lo contienen. Si alguno de estos documentos queda sin término que lo describa, entonces t se vuelve a adicionar a los documentos que lo contenían y se detiene el proceso tomando como valor de $s_{min} = s(t)$. En otro caso, si ningún documento quedó sin términos, se repite el segundo paso.

Una vez determinado el conjunto FC, en la segunda etapa se construye un conjunto inicial de grupos. Para esto, cada itemset cerrado $f_i \in FC$ forma un grupo que tiene como descripción intencional al propio f_i y como descripción extensional a $cov(f_i)$. Luego, en la tercera etapa se realiza un proceso de filtrado compuesto de dos pasos, sobre el conjunto de grupos determinados. Sea $G = \{G_i, G_2, \ldots, G_{|FC|}\}$ el conjunto de grupos determinado en la segunda etapa. Sea $L_{d_i} \subseteq G$ el conjunto de grupos que contienen a un documento $d_i \in D$. En el primer paso de este proceso de filtrado, se recorre cada documento $d_i \in D$ y se elimina dicho documento de todos los grupos en L_{d_i}, excepto de todo grupo $G \in L_{d_i}$ tal que no existe ningún otro grupo $G' \in L_{d_i}$, que cumpla que $f_G \subset f_{G'}$. En el segundo paso de este proceso de filtrado, se recorre cada documento $d_i \in D$ de la colección y se calcula el puntaje que alcanza dicho documento, para cada uno de los grupos en los que d_i está contenido. Posteriormente, se elimina d_i de todos los grupos exceptuando aquellos α grupos con los cuales alcanza el mayor puntaje; α es un parámetro del algoritmo. El puntaje de un documento d_i y un grupo $G_j \in G$, se denota por $Score(d_i, G_j)$ y se calcula de la siguiente forma:

$$Score(d_i, G_j) = \sum_{t \in f_{G_j} \cap T} w(t, d_i),$$

donde f_{G_j} es el itemset cerrado que describe intencionalmente a G_j y $w(t, d_i)$ es el peso que tiene el término t en el documento d_i.

Cuando se termina el proceso de filtrado, en la cuarta etapa se construye la jerarquía de grupos, desde abajo hacia arriba. Para esto, primero se ordena la lista de grupos resultantes de la tercera etapa, en orden

descendente de acuerdo al tamaño del itemset cerrado que los describe. Sea G la lista ordenada de grupos. Posteriormente, se recorre cada grupo $G_i \in G$ y se determina la lista L_{G_i} de grupos "padres" de G_i. L_{G_i} está formada por los grupos $G_j \in G$ que se describen intencionalmente por un itemset cerrado f_{G_j} tal que $f_{G_j} \subseteq f_{G_i}$ y $t(f_{G_i}) = t(f_{G_j}) + 1$; siendo f_{G_i} el itemset cerrado que describe intencionalmente al grupo G_i. Si $L_{G_i} = \emptyset$, entonces G_i se adiciona como nodo hijo de un nodo ficticio que se inserta como raíz de la jerarquía. En otro caso, si $L_{G_i} \neq \emptyset$, G_i se inserta como nodo hijo de cada uno de los grupos de L_{G_i}.

Una vez creada la jerarquía, en la quinta etapa se lleva a cabo un proceso para reducir el número de grupos del primer nivel de la misma. Para realizar esto se emplea una estrategia aglomerativa, similar a la del algoritmo UPGMA [9], con los grupos del primer nivel. Esta estrategia se detiene cuando el número de grupos es igual o menor a un parámetro G_{max} definido previamente. Para calcular la semejanza entre dos grupos G_1 y G_2 durante el proceso aglomerativo, se utiliza el índice Jaccard [26] sobre el conjunto de documentos contenidos en la descripción extensional de G_1 y G_2. Cuando se unen dos grupos G_1 y G_2, el grupo resultante tiene como descripción extensional e intencional, a la unión de las descripciones extensionales e intencionales de G_1 y G_2, respectivamente.

Aunque es cierto que este algoritmo no necesita que el usuario diga el umbral de soporte mínimo para el cálculo de los itemset cerrados, sí incluye dos parámetros (α y G_{max}) para la formación de la jerarquía. Estos parámetros determinan en gran medida la calidad de la jerarquía y la eficiencia del proceso de construcción de la misma. La dependencia de estos parámetros es una limitación de TCD que puede reducir su uso en problemas reales. Por otro lado, la estrategia para estimar el valor de s_{min} hace que TDC sea altamente sensible a ruido; esto es, si en la colección existe un término muy poco frecuente, entonces el proceso de estimación de s_{min} encontrará un valor muy bajo y por lo tanto, el proceso de cálculo de los itemset cerrados se volverá sumamente costoso. Por último, TDC no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

2.21.4. GPHC

GPHC [96] es un algoritmo de agrupamiento capaz de formar una jerarquía de grupos con traslape y sus descripciones, a partir de una colección de documentos. De forma similar a TDC, este algoritmo asume que cada documento está representado como un vector de términos, donde cada término tiene asociado un peso que determina la importancia del término para describir al documento; GPHC define el peso de cada término utilizando el esquema de pesado *tf-idf* [66].

GPHC basa su funcionamiento en los *conjuntos de términos cerrados e interesantes*, pero utiliza una definición completamente diferente a la que utiliza TDC para construir los itemset cerrados. Para GPHC un itemset frecuente f es cerrado si no existe otro itemset frecuente f' tal que $f' \subset f$ y $s(f') = s(f)$. Sea $f = \{t_1, t_2, \ldots, t_s\}$ un itemset cerrado. El itemset f se considera interesante de acuerdo a una medida de interés M, si el valor de la relación existente entre los conjuntos de términos $\{t_1, t_2, \ldots, t_{s-1}\}$ y $\{t_s\}$, de acuerdo a la medida M, supera un umbral previamente definido. GPHC propone seis alternativas como medidas de interés: Added value, Certainty factor, Conviction, χ^2, Mutual Information y Yules Q; la definición, las fórmulas de estas medidas, así como los valores de umbrales que propone GPHC para cada una de ellas puede consultarse en [97]. Es importante mencionar que estos valores de umbrales fueron determinados experimentalmente por GPHC.

Sea $D = \{d_1, d_2, \ldots, d_n\}$ una colección de documentos descritos por un conjunto de términos $T = \{t_1, t_2, \ldots, t_m\}$. Para formar una jerarquía a partir de esta colección, GPHC emplea una estrategia compuesta de cinco etapas. En la primera etapa, se calcula el conjunto FCI de itemsets cerrados e interesantes. Una vez determinado el conjunto FCI, en la segunda etapa se construye un conjunto inicial de grupos. Para esto, cada itemset cerrado e interesante $f_i \in FC$ forma un grupo que tiene como descripción intencional al propio f_i y como descripción extensional a $cov(f_i)$. Posteriormente, en la tercera etapa se lleva a

cabo un proceso de filtrado de este conjunto de grupos, similar al empleado por TDC. De hecho, la única diferencia radica en el segundo paso de ese proceso. A diferencia de TDC, GPHC aplica el segundo paso del proceso escalonadamente. Sea k el tamaño del itemset cerrado e interesante más grande que está incluido en FCI. En el segundo paso del proceso de filtrado que ejecuta GPHC, se recorre cada documento $d_i \in D$ y para cada valor de $p = 1 \ldots k$, se calcula el puntaje que tiene d_i para aquellos grupos G descritos por un itemset cerrado e interesante de tamaño p, en los que está contenido d_i. Luego, para cada valor de p, d_i se eliminará de todos los grupos descritos por un itemset cerrado e interesante de tamaño p a los que pertenezca, excepto a los α de mejor puntaje. La fórmula para calcular el puntaje de un documento d_i respecto a un grupo G es la misma que emplea el algoritmo TDC.

Cuando se termina el proceso de filtrado, en la cuarta etapa se construye la jerarquía de grupos, desde abajo hacia arriba. Para esto, primero se ordena la lista de grupos resultantes de la tercera etapa, en orden descendente de acuerdo al tamaño del itemset cerrado que los describe. Sea G la lista ordenada de grupos. Posteriormente, se recorre cada grupo $G_i \in G$ y se determina la lista L_{G_i} de grupos "padres" de G_i. L_{G_i} está formada por los grupos $G_j \in G$ que se describen intencionalmente por un itemset cerrado e interesante f_{G_j} tal que $f_{G_j} \subseteq f_{G_i}$ y $t(f_{G_i}) = t(f_{G_j}) + 1$; siendo f_{G_i} el itemset cerrado e interesante que describe intencionalmente al grupo G_i. Si $L_{G_i} = \emptyset$, entonces G_i se adiciona como nodo hijo de un nodo ficticio que se inserta como raíz de la jerarquía. En otro caso, si $L_{G_i} \neq \emptyset$, G_i se inserta como nodo hijo de cada uno de los grupos de $G_j \in L_{G_i}$, tal que la relación entre los conjuntos f_{G_j} y $f_{g_i} \setminus f_{G_j}$ sea interesante, de acuerdo a la medida utilizada para calcular los itemset cerrados e interesantes.

Una vez creada la jerarquía, en la quinta etapa se lleva a cabo un proceso para reducir el número de grupos del primer nivel de la misma. Para realizar esto se emplea una estrategia compuesta de tres pasos. Sea $G_{N1} = \{G_1, G_2, \ldots, G_s\}$ el conjunto de grupos del primer nivel. En el primer paso, se recorre cada grupo $G_i \in G_{N1}$ y se forma el documento ficticio d_{G_i} como la suma de todos los documentos contenidos en los grupos almacenados en el sub-árbol que define el grupo G_i en la jerarquía. Posteriormente, partiendo del grupo raíz que contiene a todos los documentos ficticios, en el segundo paso se aplica el algoritmo bisecting-Kmeans [65] hasta que cada documento ficticio forme un grupo unitario; en la aplicación del algoritmo bisecting-Kmeans se utiliza como función a optimizar el criterio I_2 [98]. Una vez terminado este paso, en el tercer paso se sustituye la aparición de cada documento ficticio d_{G_i} en los grupos determinados por el algoritmo bisecting-Kmeans, por el conjunto de objetos contenidos en el grupo G_i. De igual forma, la descripción intencional de cada uno de los nodos formados por bisecting-Kmeans se forma como la unión de las descripciones intencionales de los grupos asociados a cada documento ficticio almacenado en dicho nodo.

La principal limitación de GPHC es que requiere ajustar los valores de tres parámetros, incluido el umbral s_{min} para calcular los itemset cerrados. Por otro lado, el algoritmo no garantiza que todos los documentos de la colección pertenezcan a la jerarquía, pudiendo quedar documentos sin agrupar. Por último, GPHC no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

2.21.5. Algoritmo de Krishna y Bhavani

En [99] se propone un algoritmo de agrupamiento que, de forma similar a FTC, se basa en el concepto de itemset frecuente. Este algoritmo asume que cada documento está representado como un vector de términos, donde cada término tiene asociado un peso que determina la importancia del término para describir al documento; el peso de cada término se calcula como la frecuencia absoluta del término en el documento.

Sea $D = \{d_1, d_2, \ldots, d_n\}$ una colección de documentos descritos por un conjunto de términos $T = \{t_1, t_2, \ldots, t_m\}$. Para formar un conjunto de grupos disjuntos y sus descripciones, se emplea una estrategia compuesta de siete pasos. En el primer paso se eliminan de todo documento $d_i \in D$ aquellos términos $t_j \in T$ tal que $w(t_j, d_i) < \alpha_1$ siendo $w(t_j, d_i)$ el peso del término t_j en el documento d_i y α_1 un umbral previamente

definido. Sea $T' = \{t_1, t_2, \ldots, t_q\}$ el conjunto de términos que describen a los documentos después del paso anterior. En el segundo paso se utiliza el algoritmo Apriori [86] para calcular el conjunto F de itemset frecuentes, dado un umbral s_{min} previamente conocido. Sea k la longitud del itemset frecuente de mayor tamaño en F y $L = \{F^1, F^2, \ldots, F^k\}$ la lista de conjuntos tales que cada conjunto F^i define al conjunto de itemset frecuentes de tamaño i. En el tercer paso, la lista de itemsets contenidos en cada conjunto F^i se ordena descendentemente, de acuerdo al soporte de los itemsets. Sea $t = \frac{k}{2}$. En el cuarto paso se extrae de F^t el itemset f de mayor soporte y se forma un grupo que tiene como descripción intencional al propio f y como descripción extensional al conjunto de documentos en $cov(f)$. En el quinto paso, se elimina de la colección D el conjunto de documentos de $cov(f)$. Los pasos cuarto y quinto se repiten iterativamente mientras $F^t \neq \emptyset$ y $D \neq \emptyset$. Al terminar este proceso, si $D \neq \emptyset$, entonces en el sexto paso se decrementa el valor de t (i.e., $t = t - 1$) y se repite el proceso anterior desde el cuarto paso.

Sea $G = \{G_1, G_2, \ldots, G_k\}$ el conjunto de grupos formado producto del proceso descrito anteriormente. En el séptimo paso, se calcula para cada grupo $G_i \in G$, el conjunto de términos *clave* que describen a los documentos que forman parte de su descripción extensional. El conjunto de términos clave de G_i son aquellos términos $t_j \in T'$ que no están en el itemset que describe intencionalmente a G_i y cuya frecuencia en G_i supera un umbral α_2. Como resultado de este paso, los términos clave de cada grupo se adicionan a su descripción intencional.

Entre las limitaciones de este algoritmo está que depende de tres parámetros, incluido el umbral de soporte, el cual como se ha comentado anteriormente, determina en gran medida la eficiencia y eficacia del algoritmo. Por último, este algoritmo no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

2.21.6. F²IHC

F²IHC [100] es un algoritmo de agrupamiento que, de forma similar a FIHC, permite formar una jerarquía de grupos disjuntos y sus descripciones. No obstante, a diferencia de FIHC, F²IHC utiliza *conjuntos de términos frecuentes y difusos*, para guiar el proceso de construcción de la jerarquía y describir intencionalmente a los grupos de esta. En este contexto, un itemset es difuso si la frecuencia de cada uno de los términos que lo componen se encuentra dividida en tres regiones llamadas LOW, MID y HIGH.

Sea $D = \{d_1, d_2, \ldots, d_n\}$ una colección de documentos descritos por un conjunto de términos $T = \{t_1, t_2, \ldots, t_m\}$ del cual han sido eliminadas las *stop words*: artículos, preposiciones, adverbios, entre otras. Para formar la jerarquía F²IHC emplea una estrategia compuesta de cuatro etapas. En la primera etapa se realiza un proceso de pre-procesamiento de los documentos; para esto se utilizan cuatro pasos. En el primer paso, se recorre cada documento $d_i \in D$ y se calcula el peso que tiene cada término de $t_j \in T$, respecto a los esquemas de pesado tf-idf, tf-df y tf^2; este último se calcula como el producto del peso obtenido por los dos primeros esquemas. Posteriormente, en el segundo paso se eliminan de cada documento aquellos términos cuyos pesos en el documento, respecto a los tres esquemas empleados, no superen tres umbrales definidos previamente para cada esquema.

Sea $T' = \{t_1, t_2, \ldots, t_q\}$ el conjunto de términos que describen a los documentos luego del segundo paso. En el tercer paso, se realiza una descomposición del peso de cada término $t_j \in T'$ que describe a los documentos de la colección. Para esto, se recorre cada documento $d_i \in D$ y se calcula, para cada término $t_j \in T'$, el peso de t_j en d_i, para las regiones LOW, MID y HIGH. Sea f_{ij} la frecuencia absoluta del término t_j en el documento d_i. Los pesos de t_j en d_i, respecto a las regiones LOW, MID y HIGH, denotados como

w_{ij}^L, w_{ij}^M y w_{ij}^H, respectivamente, se calculan de la siguiente forma:

$$w_{ij}^L = \begin{cases} 0, & \text{si } f_{ij} = 0 \\ 1 + f_{ij}, & \text{si } 0 < f_{ij} < a_1 \\ 2, & \text{si } f_{ij} = a_1 \\ 1 + a_2, -\frac{f_{ij}}{a_2} - a_1, & \text{si } a_1 < f_{ij} < a_2 \\ 1, & \text{si } f_{ij} \geq a_2 \end{cases},$$

$$w_{ij}^M = \begin{cases} 0, & \text{si } f_{ij} = 0 \\ 1, & \text{si } f_{ij} = a_1 \\ 1 + f_{ij} - \frac{a_1}{a_2} - a_1, & \text{si } a_1 < f_{ij} < a_2 \\ 2, & \text{si } f_{ij} = a_2 \\ 1 + a_3 - \frac{f_{ij}}{a_3} - a_2, & \text{si } 2_2 < f_{ij} < a_3 \\ 1, & \text{si } f_{ij} = a_3 \end{cases},$$

$$w_{ij}^H = \begin{cases} 0, & \text{si } f_{ij} = 0 \\ 1, & \text{si } f_{ij} \leq a_1 \\ 1 + \frac{f_{ij}}{a_2} - a_1, & \text{si } a_1 < f_{ij} < a_2 \\ 2, & \text{si } f_{ij} = a_2 \end{cases},$$

donde $a_1 = min\{f_{ij}\}$ y $a_3 = max\{f_{ij}\}$ representan la menor y mayor frecuencia del término $t_j \in T'$ en toda la colección, respectivamente; $a_1 = ave\{f_{ij}\}$ es el promedio de frecuencias distintas de cero que tiene el término $t_j \in T'$ en toda la colección.

Una vez terminado el paso anterior, en el cuarto paso se calcula el peso total que tiene cada término $t_j \in T'$ respecto a las regiones LOW, MID y HIGH, denotados como w_j^L, w_j^M y w_j^H, respectivamente. Cuando termina este cálculo, se determina para cada término $t_j \in T'$ la región de mayor peso, denotada por $maxReg_{t_j}$. Los fórmulas para calcular w_j^L, w_j^M y w_j^H son las siguientes:

$$w_j^L = \sum_{i=1}^n w_{ij}^L, \qquad w_j^M = \sum_{i=1}^n w_{ij}^M, \qquad w_j^H = \sum_{i=1}^n w_{ij}^H,$$

En la segunda etapa se calcula, dado un valor de umbral s_{min}, el conjunto F de itemset frecuentes y difusos. Para realizar esto, F^{2}IHC define los conceptos de soporte de un itemset difuso.

Definición 20 (soporte de un itemset difuso). *El soporte de un itemset difuso $f \subseteq T$, de tamaño $k \geq 1$, se denota como $sd(f)$ y se calcula como $sd(f) = \sum_{i=1}^n Min\{w_{ij}^{maxReg_{t_j}} \mid t_j \in f\}$; donde t_j es un término que pertenece al itemset difuso f, y $w_{ij}^{maxReg_{t_j}}$ es el peso que tiene, en el documento i, el término t_j en su mejor región.*

Para calcular el conjunto F de itemset frecuentes y difusos, F^{2}IHC utiliza una variante del algoritmo Apriori [86], pero utilizando la definición de soporte descrita anteriormente. Sea $F_1 = F \cap T'$ el conjunto de itemset frecuentes y difusos, de tamaño 1 y $F_{\geq 1} = F \setminus F_1$ el resto de los itemset frecuentes y difusos determinados en la segunda etapa. En la tercera etapa se realiza un proceso de filtrado de los itemset del conjunto $F_{\geq 1}$. Para esto, se recorre cada itemset $f \in F_{\geq 1}$ y calcula la *confianza* de todas las *reglas de asociación* que se pueden extraer del itemset f [86]. Una regla de asociación es una implicación $A \rightarrow B$, en la cual A y B son conjuntos de términos tales que $A \cap B = \emptyset$; A se conoce como *antecedente* y B como *consecuente*. La confianza de una regla de asociación $A \rightarrow B$ se calcula como la razón entre el soporte del conjunto $A \cup B$ y el soporte del conjunto A. La confianza expresa la probabilidad condicional del consecuente de la regla respecto a su antecedente. Si la confianza de alguna regla de asociación formada

a partir del itemset $f \in F_{\geq 1}$, supera un umbral c_{min} previamente definido, entonces se adiciona el itemset f a un conjunto F'. El conjunto de itemset frecuentes y difusos que utiliza F^{2}IHC para formar la jerarquía se denota por CF y está compuesto por la unión entre F_1 y F'.

En la tercera etapa se construye la jerarquía, a partir del conjunto CF y utilizando ocho pasos. En el primer paso, se forma con cada itemset $f \in CF$ un grupo que tiene como descripción intencional al propio f. Sea $G = \{G_1, G_2, \ldots, G_k\}$ el conjunto de grupos formados en el primer paso. En el segundo paso, se forma la matriz V_{n*k}, donde n es el número de documentos y k el número de itemset en el conjunto CF; esta matriz contiene el grado de pertenencia de cada documento a cada grupo. La matriz V_{n*k} se obtiene a través de la multiplicación de las matrices W_{n*m} y Q_{m*k}, donde m es el número de términos en el conjunto T'. La matriz W_{n*m} contiene en cada celda w_{ij} el valor $w_{ij}^{maxReg_{t_j}}$; es decir, el peso que tiene en su mejor región, cada término $t_j \in T'$ en el documento $d_i \in D$. Por otra parte, la matriz Q_{m*k} contiene en cada celda q_{jt}, el grado de importancia que tiene el término $t_j 1 T'$ para el grupo G_t. En el tercer paso, se visita cada documento $d_i \in D$ de la colección y se asigna dicho documento al grupo descrito intencionalmente por un itemset frecuente y difuso de tamaño 1, con el cual d_i alcanza el mayor grado de pertenencia, de acuerdo a la matriz V_{n*k}, de todos los grupos descritos por un itemset frecuente y difuso de tamaño 1. Si d_i alcanza el máximo para más de un grupo, entonces d_i se asigna al grupo candidato cuyo itemset tiene el mayor soporte. Posteriormente, en el quinto paso se elimina de G todo grupo descrito por un itemset frecuente y difuso de tamaño 1, cuya descripción extensional esté vacía.

Sea $G^k \subseteq G$ el conjunto de grupos descritos por un itemset frecuente y difuso de tamaño k. En el sexto paso se procesa el conjunto G^1, uniendo iterativamente los pares de grupos cuya semejanza sea mayor a un umbral previamente establecido. Sea $G_1, G_2 \in G^1$ dos grupos que se van a unir. Asúmase, sin pérdida de generalidad, que la descripción extensional de G_1 es mayor que la de G_2. Unir estos dos grupos significa insertar en la descripción extensional del grupo G_1 aquellos documentos de la descripción extensional de G_2 y eliminar al grupo G_2. Este proceso se repite mientras la semejanza entre algún par de grupos supere el umbral establecido. Cada vez que se unen dos grupos hay que modificar las matrices W, Q y V. El conjunto de grupos de G^1 resultante de este paso, constituye el nivel tope de la jerarquía. Posteriormente, en el séptimo paso se elimina del conjunto de grupos G todo aquel grupo $G' \in G^2$ tal que no exista grupo $G'' \in G^2$ que cumpla que $f_{G''} \subset f_{G'}$; siendo $f_{G''}$ y $f_{G'}$ los itemset frecuentes y difusos que describen intencionalmente a los grupos G'' y G', respectivamente. Para formar el nivel $i > 1$, en el octavo paso se recorren los grupos del nivel i y para cada grupo G' del nivel i, se inserta en la lista de nodos hijos de G' todo aquel grupo $G'' \in G^{i+1}$ tal que $f_{G'} \subset f_{G''}$; siendo $f_{G'}$ y $f_{G''}$ los itemsets frecuentes y difusos que describen intencionalmente a los grupos G' y G'', respectivamente. Los pasos séptimo y octavo se repiten iterativamente hasta que se procesen todos los grupos de G.

Cuando se termina de formar la jerarquía, en la cuarta etapa se visita cada nodo de la misma. Durante este recorrido, en cada nodo se compara el índice de pertenencia que tienen los documentos contenidos en dicho nodo, utilizando la matriz V, con respecto al índice de pertenencia que tienen los mismos documentos en los nodos hijos. Si algún documento tiene mayor índice de pertenencia en un nodo hijo que en el nodo actual, entonces este documento se elimina de la descripción extensional del grupo que representa al nodo actual y es insertado en la descripción extensional del grupo que representa a dicho nodo hijo.

La principal limitación de este algoritmo es su alta complejidad computacional, debido al proceso de construcción de los itemsets frecuentes y difusos, y al mismo proceso de construcción de la jerarquía; este aspecto reduce su aplicación en problemas reales. Adicionalmente, F^{2}IHC necesita ajustar los valores de más de seis parámetros, lo cual igual dificulta su aplicación. Otro aspecto negativo es que el proceso de descomposición de los pesos de los términos, así como el proceso de mantenimiento de las matrices V, W y Q hacen que el algoritmo consuma mucha memoria. Por último, F^{2}IHC no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

2.21.7. IDHC

IDHC [101,102] es un algoritmo de agrupamiento que, dada una colección de documentos, permite formar una jerarquía de grupos con traslape y sus descripciones. Este algoritmo asume que cada documento está representado como un vector de términos, donde cada término tiene asociado un peso que determina la importancia del mismo para describir al documento; IDHC define el peso de cada término como la frecuencia absoluta del término dentro del documento.

Sea $D = \{d_1, d_2, \ldots, d_n\}$ una colección de documentos descritos por un conjunto de términos $T = \{t_1, t_2, \ldots, t_m\}$. Para formar la jerarquía de grupos, IDHC emplea una estrategia compuesta de dos etapas. En la primera etapa, se realiza un proceso de selección de términos. Para realizar este proceso se emplean dos pasos. En el primer paso, IDHC selecciona del conjunto T, aquellos términos que estén contenidos en no más de $MaxDoc$ y en no menos de $MinDoc$ documentos, eliminando el resto de los términos de los documentos de la colección; $MaxDoc$ y $MinDoc$ son parámetros del algoritmo. Posteriormente, en el segundo paso se recorre cada documento de la colección y se elimina todo término del mismo, excepto los k términos de mayor frecuencia; k es un parámetro del algoritmo.

Una vez que se reduce la dimensionalidad de los documentos, en la segunda etapa IDHC selecciona de cada documento aquellos pares de términos que sean *significativos*. Para esto, en cada documento $d_i \in D$ se ejecutan dos pasos. En el primer paso, se calcula la *significancia* de todos los pares de términos incluidos en d_i. La significancia de un par de términos $\{t_j, t_q\}$ se denota por $Sig(\{t_j, t_q\})$ y se calcula como el producto entre la significancia local y global del par $\{t_j, t_q\}$. La significancia local de $\{t_j, t_q\}$ es el promedio de los pesos de cada término en d_i. La significancia global de $\{t_j, t_q\}$ se calcula, de la misma forma y con las mismas medidas, en que el algoritmo GPHC calcula el interés de un itemset. Sea L_{d_i} la lista de pares de términos incluidos en el documento d_i. Posteriormente, en el segundo paso se ordena L_{d_i}, descendentemente por el valor de significancia de los pares de términos, y se calcula la media y la desviación estándar de dichos valores. Sea P_{d_i} el conjunto de a lo sumo $maxK$ pares de términos de d_i que tienen el mayor valor de significancia; $maxK$ es un parámetro del algoritmo. Los pares de términos *significativos* de d_i son aquellos pares incluidos en P_{d_i} cuya significancia supere un umbral α, previamente establecido.

Sea PT_{sig} el conjunto de pares de términos significativos, extraídos de todos los documentos de D. En la tercera etapa, se crea con cada par de términos $p_i \in PT_{sig}$ un grupo que tiene como descripción intencional al propio p_i y como descripción extensional a los documentos de D que contienen a dicho par. Sea G' el conjunto inicial de grupos. Posteriormente, se lleva a cabo un proceso que tiene como objetivo unificar a aquellos grupos que tengan la misma descripción extensional, pero difieran en cuanto a su descripción intencional. Si se tienen que unir dos o más grupos, el grupo resultante será aquel que tiene la misma descripción extensional que los grupos a unir, pero que tiene como descripción intencional a la unión de las descripciones intencionales de todos los grupos a unir.

Sea $G = \{G_1, G_2, \ldots, G_k\}$ el conjunto de grupos resultantes de la tercera etapa. En la cuarta etapa, se utiliza el conjunto G para formar una jerarquía de grupos, de arriba hacia abajo, a través de un proceso de refinamiento iterativo. El nivel tope de la jerarquía está formado por los grupos del conjunto G. A partir de este punto, cada nivel siguiente es construido utilizando el nivel anterior. Para construir el nivel $i > 1$, se utilizan dos pasos. En el primer paso se detectan en el nivel $i-1$ todos los pares de grupos no unitarios G_1, G_2 tales que la intersección entre sus descripciones extensionales es distinta del vacío. Para estos pares de grupos, se crea un nuevo grupo que tiene como descripción extensional la intersección entre las descripciones extensionales de G_1 y G_2, y como descripción intencional, la unión de las descripciones intencionales de G_1 y G_2. En el segundo paso, se procesa el conjunto de grupos formados en el primer paso, utilizando la misma estrategia de unificación de la tercera etapa. El conjunto de grupos resultante

de este paso constituye el agrupamiento del nivel i. Los dos pasos anteriores se repiten mientras se sigan generando niveles.

Una vez terminada la construcción de la jerarquía, se lleva a cabo una quinta etapa que tiene como objetivo reducir el número de grupos del nivel tope de la jerarquía. Para realizar esto se emplea la misma estrategia que utiliza el algoritmo GPHC en su quinta etapa.

La principal limitación de IDHC es que necesita ajustar valores para cinco parámetros, que dependen de la colección a procesar. Esta limitación dificulta la aplicación de este algoritmo en problemas reales. Por otro lado, la estrategia de construcción de la jerarquía que emplea IDHC puede generar grupos vacíos dentro de la misma. Por último, GPHC no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

2.21.8. Algoritmo de Kiran et al.

En [103] se propone un algoritmo de agrupamiento que se basa en algunas de las ideas propuestas por los algoritmos TDC [94] y GPHC [96]. Este algoritmo permite formar una jerarquía de grupos con traslape y las descripciones de éstos, a partir de una colección de documentos. El algoritmo asume que cada documento está representado como un vector de términos que tienen asociado un peso que expresa la importancia del término para describir al documento; el peso se determina utilizando el esquema de pesado *tf-idf* [66].

Sea $D = \{d_1, d_2, \ldots, d_n\}$ una colección de documentos descritos por un conjunto de términos $T = \{t_1, t_2, \ldots, t_m\}$. Para formar la jerarquía, este algoritmo utiliza una estrategia compuesta de cinco etapas. En la primera etapa se calcula el conjunto F de itemset frecuentes, utilizando el algoritmo Apriori [86] y un umbral de soporte s_{min}. El valor de s_{min} se estima utilizando la misma estrategia que el algoritmo TDC (ver subsección 2.21.3). Posteriormente, en la segunda etapa se utiliza un umbral ε y el conjunto F para determinar el conjunto F' de conjuntos de términos frecuentes generalizados [104]. F' es un subconjunto del conjunto de términos cerrados de una colección, a partir del cual se puede estimar el soporte de todos los itemset frecuentes de la colección, cometiendo a lo sumo un error igual a ε en dicho cálculo; siendo ε un umbral previamente definido.

Usando el conjunto F', en la tercera etapa se crea un conjunto de grupos iniciales G. Para esto, se forma un grupo con cada itemset $f_i \in F'$, que tiene como descripción intencional a f_i y como descripción extensional a los documentos de $cov(f_i)$. Posteriormente, en la cuarta etapa se realiza un proceso de filtrado que tiene como objetivo limitar el número de grupos a los que puede pertenecer un documento. Para este propósito el algoritmo propone dos alternativas. La primera es la misma que emplea TDC, en el segundo paso del proceso de filtrado que ejecuta en la tercera etapa. No obstante, este algoritmo modifica ligeramente la fórmula para calcular el puntaje de un documento d_i respecto a un grupo G_j. La nueva fórmula es:

$$Score(d_i, G_j) = \frac{\sum_{t \in f_{G_j} \cap T} w(t, d_i)}{\left|G_j\right|},$$

donde f_{G_j} es el itemset que describe intencionalmente al grupo G_j y $w(t, d_i)$ es el peso que tiene el término t_j en el documento d_i.

La otra alternativa es más compleja y consume muchas más memoria. Esta alternativa consiste en utilizar Wikipedia para enriquecer el contenido de los documentos y posteriormente, decidir a cuántos grupos puede pertenecer cada documento. En el primer paso de esta estrategia, se construye para cada documento $d_i \in D$, los vectores d_i^{CAT} y d_i^{LINK}. Para construir el vector d_i^{CAT} se buscan en Wikipedia, todas las páginas relacionadas con los términos $t_j \in T$ que describen a d_i y se toman las categorías de dichas páginas. El vector d_i^{CAT} está formado por las categorías extraídas para d_i, en el paso anterior, y un peso

asociado a cada categoría que determina la frecuencia absoluta de dicha categoría; i.e., el número de veces que se encontró cada categoría en el primer paso. El vector d_i^{LINK} se construye de forma similar al vector d_i^{CAT}, pero utilizando los *outlinks* de las páginas de Wikipedia relacionadas con cada término. Un *outlink* es una palabra de una página de Wikipedia, que tiene un enlace a otra página. Utilizando estos vectores, la fórmula para calcular el puntaje de un documento d_i respecto a un grupo G_j se define como:

$$Score(d_i, G_j) = \frac{\sum_{d_t \in G_j} Sim(d_i, d_j)}{|G_j|},$$

donde $Sim(d_i, d_j)$ es la semejanza entre el documento d_i y los documentos d_j, que pertenecen a la descripción extensional del grupo G_j. En este algoritmo, la semejanza entre dos documentos se calcula como sigue:

$$Sim(d_i, d_j) = cos(d_i, d_j) + \alpha \cdot cos(d_i^{CAT}, d_j^{CAT}) + beta \cdot cos(d_i^{LINK}, d_j^{LINK}),$$

siendo $cos(d_i, d_j)$, $cos(d_i^{CAT}, d_j^{CAT})$ y $cos(d_i^{LINK}, d_j^{LINK})$ la semejanza existente entre los documentos d_i y d_j, respecto a los vectores de términos, categorías y outlinks que representan al documento d_i, respectivamente.

Una vez terminada la cuarta etapa, en la quinta se construye una jerarquía, utilizando la misma estrategia que emplea el algoritmo GPHC (cuarta etapa). Cuando ya está creada la jerarquía, ésta se recorre y se modifica la descripción intencional de cada grupo, añadiéndole las k categorías de Wikipedia más frecuentes en los documentos que forman parte de la descripción extensional de dicho grupo.

La principal limitación de este algoritmo es su alta complejidad computacional, inherente al proceso de construcción de la jerarquía, y su alto consumo de memoria, lo cual incide negativamente en su uso en problemas reales. Otra limitación es que depende de cuatro parámetros que dependen de la colección a procesar. Por otro lado, el algoritmo puede construir descripciones que sean muy largas y por lo tanto, difíciles de interpretar. Por último, este algoritmo no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

2.21.9. CFWS y CFWMS

CFWS y CFWMS son algoritmos de agrupamiento que permiten formar un conjunto de grupos, en los cuales es permitido el traslape y donde cada grupo cuenta con una descripción extensional y otra intencional [105]. A diferencia de algoritmos como FTC, FIHC y TDC entre otros, que utilizan itemset frecuentes para guiar el proceso de agrupamiento, estos algoritmos están basados en secuencias frecuentes. Una secuencia es un conjunto de términos (en el sentido general de la palabra) que ocurren en un determinado orden. El resto de los conceptos y definiciones establecidos para itemsets, se aplican también para secuencias.

Sea $D = \{d_1, d_2, \ldots, d_n\}$ una colección de documentos descritos por secuencias de términos del conjunto $T = \{t_1, t_2, \ldots, t_m\}$. Para agrupar la colección D, CFWS trabaja con secuencias de palabras. Por otra parte, CFWMS sustituye cada término incluido en los documentos por los conceptos de WordNet que se relacionan y trabaja con secuencias de conceptos frecuentes.

Para formar el agrupamiento, CFWS utiliza una estrategia formada de cinco etapas. En la primera etapa los documentos son procesados y transformados para poder ser utilizados en el resto de las etapas. Para lograr este propósito se emplean dos pasos. En el primer paso, se determinan todos los pares de términos que están contenidos en al menos α documentos de la colección; α es un parámetro del algoritmo. Posteriormente, en el segundo paso se eliminan de todos los documentos aquellos términos que no pertenezcan a ninguno de los pares de términos seleccionados en el primer paso.

En la segunda etapa, se determinan las secuencias de términos frecuentes (en lo siguiente referidas como secuencias frecuentes) de la colección. Para esto se utilizan dos pasos. En el primer paso, se construye

un árbol de sufijos generalizados, que contiene a todos los sufijos de los documentos de la colección. Un árbol de sufijos generalizado es un árbol dirigido en el cual existen dos tipos de nodos: internos y hojas. En este árbol, las aristas están etiquetadas con una subcadena no vacía de una cadena; las etiquetas de los nodos es la concatenación de las etiquetas de las aristas que se encuentran en el camino desde la raíz al nodo. Por último, en cada nodo hay almacenado una lista que contiene a todos los documentos de la colección, que contienen a la secuencia s que etiqueta a dicho nodo, exceptuando a todo aquel documento que contenga una secuencia $s \subseteq s'$. Para calcular las secuencias frecuentes, se recorre el árbol de sufijos en profundidad, teniendo en cuenta que el soporte de una secuencia está dado por la cantidad de documentos almacenados en el sub-árbol que define el nodo que está etiquetado con dicha secuencia. Toda secuencia cuyo soporte exceda un umbral s_{min} se considera frecuente.

Sea SF el conjunto de secuencias frecuentes. En la tercera etapa se forma por cada secuencia $s_i \in SF$, un grupo que tiene como descripción intencional a s_i y como descripción extensional a $cov(s_i)$. Sea $G = \{G_1, G_2, \ldots, G_k\}$ el conjunto de grupos determinados en la etapa anterior. En la cuarta etapa se lleva a cabo un proceso de cuatro pasos, en el cual se unen los grupos $G_i, G_j \in G$ cuyas descripciones intencionales (i.e., secuencias) *cotejen a un nivel z*. Para esto, en el primer paso se ordenan los grupos del conjunto G descendentemente, de acuerdo al tamaño de la secuencia frecuente que los describe. En el segundo paso se recorre cada grupo $G_i \in G$ y se determina el conjunto de grupos $L_{G_i} = \{G_1^{'}, G_2^{'}, \ldots, G_s^{'}\}$ cuyas descripciones intencionales cotejen con la descripción intencional de G_i a un nivel z. Dos secuencias s y t cotejan a nivel z, si existe una subsecuencia $t' \subset t$, de tamaño $|s|$ que tiene $|s| - z$ términos en común con la secuencia s. En el tercer paso se crea un grupo G_i^* que tiene como descripción intencional e extensional a la unión de las descripciones intencionales e extensionales de G_i y los grupos de L_{G_i}. En el cuarto paso se inserta G_i^* en un conjunto G' y se eliminan de G el grupo G_i y los grupos de L_{G_i}.

Por último, en la quinta etapa se realiza un proceso que tiene como objetivo reducir aún más, el número de grupos del conjunto G'. Para esto, iterativamente se van uniendo los grupos G_i, G_j cuyas descripciones extensionales tengan un traslape mayor que un umbral t_{max}. Este proceso se repite hasta que el traslape de ningún par de grupos supere el umbral t_{max} o, hasta que el número de grupos alcanzados sea igual a un umbral g_{max}. Cuando se unen dos grupos, el resultado es un grupo que tiene como descripción intencional e extensional a la unión de las descripciones intencionales e extensionales de dichos grupos, respectivamente.

Entre las limitaciones de CFWS se encuentra el consumo de memoria que tiene el árbol de sufijos generalizado. Cuando el número de documentos de la colección es grande y el umbral α es bajo, el mantenimiento de este árbol resulta extremadamente costoso. Por otro lado, el algoritmo necesita ajustar valores de cinco parámetros, incluidos el umbral de soporte s_{min}. Adicionalmente, los procesos de reducción de grupos que se realizan en la etapa cuatro y cinco, hacen que las descripciones de los grupos sean grandes y por lo tanto, que sea difícil su interpretación. Por último, este algoritmo no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

Para construir el agrupamiento, el algoritmo CFWMS utiliza una estrategia compuesta de cuatro etapas. En la primera etapa se eliminan de los documentos los adjetivos y adverbios. Posteriormente, cada sustantivo y verbo es sustituido por una lista *MU* que contiene: i) los dos conceptos de WordNet que se asocian con los dos significados más frecuentes de dichos sustantivos y verbos, ii) los sinónimos de dichos conceptos de WordNet. Posteriormente, se buscan los pares de listas *MU* que son frecuentes, utilizando cotejo inexacto para el cálculo de su soporte. Una vez determinados los pares de listas *MU* frecuentes, se elimina de cada documento aquellas listas *MU* que no pertenezcan a ningún par de listas *MU* frecuente. A partir de este punto, las etapas dos, tres y cuatro que realiza el algoritmo CFWMS coinciden con las etapas dos, tres y cinco de CFWS, anteriormente descritas. Las limitaciones de CFWMS son las mismas que las del algoritmo CFWS.

2.21.10. MC

El algoritmo MC [106] permite formar un conjunto de grupos disjuntos y sus descripciones, a partir de una colección de documentos, descritos por un conjunto de términos.

Para construir el agrupamiento conceptual, MC utiliza una estrategia compuesta de doce pasos. En el primer paso, se calcula el conjunto F de itemset frecuentes de la colección, utilizando el algoritmo Apriori [86]. En el segundo paso cada documento se representa como un vector de los itemset frecuentes que contiene, donde cada itemset tiene un peso asociado que es el soporte de dicho itemset en la colección. En el tercer paso, se construye una matriz de semejanza M que refleja la semejanza que existe entre todos los pares de documentos de la colección. Para calcular la semejanza entre un par de documentos MC propone tres alternativas: i) el número de itemset frecuentes que tienen en común los documentos, ii) el peso total de todos los itemset frecuentes y iii) la semejanza binaria asimétrica entre los itemset frecuentes que describen a los documentos.

Sea *min* el menor valor de semejanza, distinto de cero, que está almacenado en M. En el cuarto paso, se selecciona el mayor valor de semejanza, denotado como *max*, que se encuentra en M. Sea i, j la fila y columna asociada al valor *max*. Si $max = min$, entonces se verifica si los documentos d_i y d_j, cuya semejanza es igual a *max*, pertenecen a algún grupo. Si ninguno de los dos pertenece a ningún grupo, entonces se crea un grupo nuevo que contiene a ambos documentos en su descripción extensional. En otro caso, si uno de los dos documentos pertenece a un grupo G, entonces se crea un nuevo grupo que contiene en su descripción extensional al otro documento. Por el contrario, si $max \neq min$, entonces si tanto d_i como d_j no pertenecen a ningún grupo, se crea un grupo nuevo que contiene a ambos documentos en su descripción extensional. Por el contrario, si uno de los dos pertenece a un grupo G, entonces el otro documento también se inserta en la descripción extensional de G. En el quinto paso se asigna en la posición i, j de M, el valor 0. Los pasos cuatro y cinco se repiten hasta que $max = 0$. Cuando termina este proceso, si aún quedan documentos que no pertenecen a ningún grupo, entonces en el sexto paso se crea un grupo, para cada uno de estos documentos, que tiene en su descripción extensional a dicho documento.

Sea $G = \{G_1, G_2, \ldots, G_k\}$ el conjunto de grupos determinado hasta el momento. En el séptimo paso, para cada grupo $G_i \in G$ se selecciona el itemset que tenga el mayor soporte y mayor tamaño, dentro de todos los itemset frecuentes que describen a un documento contenido en el grupo. El itemset seleccionado pasa a ser la descripción intencional de G_i. En el octavo paso se recorre cada grupo $G_i \in G$ y se verifica si su descripción intencional es la misma que la de algún otro grupo. Si esto sucede, entonces se busca dentro de G_i el próximo itemset que tenga el mayor soporte y mayor tamaño, dentro de todos los itemset frecuentes que describen a un documento contenido en el grupo y que además, no esté contenido en la descripción intencional del resto de los grupos; este otro itemset pasa a ser la nueva descripción intencional de G_i.

En el noveno paso se inicializa a cero el índice de creación de todos los grupos de G y en el décimo paso se calcula la semejanza entre todos los pares de grupos $G_i, G_j \in G$, que tengan el índice de creación más bajo. Si todas estas semejanzas son cero, entonces se termina el algoritmo y el conjunto de grupos determinado hasta ese momento constituye el agrupamiento final. En otro caso, en el onceno paso se selecciona la mayor semejanza y se unen los grupos G_i, G_j que alcanzan dicha semejanza. La unión de dos grupos G_i, G_j forma un nuevo grupo G_{ij} que tiene como descripción extensional e intencional, a la unión de las descripciones extensionales e intencionales de G_i y G_j, respectivamente. En el paso número doce se elimina de G a los grupos G_i, G_j y en su lugar se inserta el nuevo grupo G_{ij}, con índice de creación igual a la suma de los índices de creación de G_i y G_j, más uno. Los pasos diez, once y doce se repiten hasta alcanzar la condición de parada. La semejanza entre un par de grupos G_i, G_j se calcula como la suma de las semejanzas que existen entre todo par de documentos $d_t \in G_i$ y $d_q \in G_j$.

La mayor limitación de este algoritmo es su consumo de memoria; cuando el número de documentos crece, el mantenimiento de la matriz de semejanza M se hace ineficiente. Por otro lado, MC depende

del parámetro s_{min}. Como se ha mencionado anteriormente, valores muy bajos hacen al algoritmo muy costoso, mientras que valores muy altos pueden formar agrupamientos de baja calidad, al no descubrir conjuntos de términos que puedan ser interesantes. Por último, este algoritmo no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

2.21.11. CPC

CPC [107] es un algoritmo de agrupamiento que, dada una colección de objetos, permite formar un conjunto de grupos disjuntos y sus descripciones. CPC no necesita de una medida de distancia entre los objetos y basa su estrategia de agrupamiento en los *conjuntos de términos frecuentes contrastantes*. Un itemset frecuente es contrastante si tiene un soporte alto en un conjunto de objetos y un soporte bajo en el resto de la colección. Este algoritmo asume que los objetos de la colección se describen por rasgos categóricos. Para presentar el algoritmo CPC es necesario introducir primero algunos conceptos.

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos descritos por un conjunto de atributos cualitativos y F, el conjunto de itemsets frecuentes en O para un umbral de soporte s_{min}, previamente establecido.

Definición 21 (itemset *mutuo* de dos itemsets). *Un itemset p' es un itemset* mutuo *de dos itemsets p_1, p_2, si y solo si se cumple que $cov(p') \cap cov(p_1) \neq \emptyset$ y $cov(p') \cap cov(p_2) \neq \emptyset$.*

La medida MPQ entre dos itemsets p_1 y p_2, denotado por $MPQ(p_1, p_2)$, estima la coherencia total que existe entre dichos itemsets, a través de la coherencia que les aporta cada uno de sus itemsets *mutuos*. Mientras mayor sea el valor $MPQ(p_1, p_2)$ más coherentes serán dichos itemsets y por lo tanto, debieran pertenecer a un mismo grupo. Por el contrario, si el valor es bajo, entonces dichos itemsets deben pertenecer a grupos diferentes. La fórmula para calcular $MPQ(p_1, p_2)$ es la siguiente:

$$MPQ(p_1, p_2) = \frac{PQ2(p_1, p_2)}{PQ1(p_1) \cdot PQ1(p_2)},$$

donde $PQ2(p_1, p_2)$ denota la coherencia preliminar que hay entre los itemsets p_1 y p_2; $PQ1(p_1)$ y $PQ1(p_2)$ denotan la calidad de los itemsets p_1 y p_2, respecto al traslape con otros itemsets.

Sea $F_1 = F \setminus \{p_1, p_2\}$. La fórmula para calcular $PQ2(p_1, p_2)$ es la siguiente:

$$PQ2(p_1, p_2) = \sum_{x \in F_1} \left(\frac{|cov(p_1) \cap cov(x)| \cdot |cov(p_2) \cap cov(x)|}{|cov(x)|} \cdot \left[\frac{|x_{max}|}{mgLen(x)} \right]^2 \right),$$

donde x_{max} denota al itemset frecuente y cerrado asociado al itemset x; $mgLen(x)$ denota el tamaño promedio de los itemsets mínimos respecto al operador $\subseteq$, que están en la clase de equivalencia del itemset x. La clase de equivalencia de un itemset f está compuesta de todo aquel itemset f' tal que $cov(f) = cov(f')$.

Sea $F_2 = F \setminus \{p_1\}$. La fórmula para calcular la medida $PQ1$ para un itemset p_1 es la siguiente:

$$PQ1(p_1) = \sum_{x \in F_2} \left(|cov(x) \cap cov(p_1)| \cdot \left[\frac{|x_{max}|}{mgLen(x)} \right]^2 \right).$$

La medida MPQ también está definida para un itemset p_1 y un conjunto de itemsets PS, de la siguiente forma:

$$MPQ(p_1, PS) = \frac{PQ2(p_1, PS)}{PQ1(p_1) \cdot PQ1(PS)},$$

estando definidas $PQ2(p_1, PS)$ y $PQ1(PS)$ de la siguiente forma:

$$PQ2(p_1,p_2) = \sum_{x \in F_3} \left(\frac{|cov(p_1) \cap cov(x)| \cdot |cov(PS) \cap cov(x)|}{|cov(x)|} \cdot \left[\frac{|x_{max}|}{mgLen(x)} \right]^2 \right),$$

y

$$PQ1(PS) = \sum_{x \in F_4} \left(|cov(x) \cap cov(PS)| \cdot \left[\frac{|x_{max}|}{mgLen(x)} \right]^2 \right),$$

siendo $F_3 = F \setminus (\{p_1\} \cup PS)$ y $F_4 = F \setminus PS$.

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos descritos por un conjunto de atributos cualitativos. Para formar el agrupamiento, CPC emplea una estrategia compuesta de cinco etapas. En la primera etapa, se utiliza el algoritmo FP-growth [108] para calcular el conjunto de itemsets frecuentes de la colección, dado un umbral de soporte s_{min}. En la segunda etapa, se selecciona del conjunto F, un subconjunto de k itemsets "semillas" $S = \{p_1, p_2, \ldots, p_k\}$ tal que $\sum_{p_i,p_j \in S} MPQ(p_i, p_j)$ sea baja. Esta forma de seleccionar a las semillas permite que la selección de S sea óptima, al garantizar que el valor de MPQ entre todos los pares de itemsets del conjunto S es bajo. Para obtener el conjunto S se sigue una estrategia compuesta de tres pasos. En el primer paso, se crean M conjuntos $S_1, S_2, \ldots, S_M$, cada uno conteniendo k itemsets frecuentes del conjunto F. Los itemsets p seleccionados para cada conjunto S_i deben cumplir dos condicione: i) tener un soporte igual o mayor al promedio de soporte de todos los itemsets del conjunto F y ii) para cualquier otro itemset p' contenido en S_i, debe cumplirse que $|cov(p) \cap cov(p')| \leq \alpha \cdot min(|cov(p)|, |cov(p')|)$. Una vez formados los M conjuntos, en el segundo paso se seleccionan los N conjuntos que minimicen $\sum_{p_i,p_j \in S} MPQ(p_i, p_j)$ y en el tercer paso, se lleva a cabo un proceso de refinamiento con cada uno. Luego de este proceso de refinamiento, el conjunto S que minimice $\sum_{p_i,p_j \in S} MPQ(p_i, p_j)$ es el conjunto de itemsets "semilla".

Para refinar un conjunto S de k itemsets frecuentes, se realiza un proceso compuesto de tres pasos. En el primer paso se selecciona el par de itemsets $p, q \in S$ que maximicen la medida $MPQ(p,q)$ dentro del conjunto S. Posteriormente, en el segundo paso se buscan los patrones $p', q' \in F \setminus S$ que mejor reemplacen a los itemsets p y q, respectivamente. Un itemset p puede reemplazar a un itemset p en el conjunto S, si se cumple que p' satisface las condiciones i) y ii), y además, minimiza la medida $MPQ(p', t)$ para todo otro itemset $t \neq p$ del conjunto S. El itemset que mejor reemplaza a otro será aquel que además de lo anteriormente comentado, minimice $\sum_{p_i,p_j \in S} MPQ(p_i, p_j)$. Una vez que se tienen los itemsets p' y q', en el tercer paso se decide reemplazar a p o a q en dependencia de cual acción minimice $\sum_{p_i,p_j \in S} MPQ(p_i, p_j)$. El proceso anterior se repite desde el primer paso, mientras se encuentre reemplazo para algún itemset de S.

Sea $S = \{p_1, p_2, \ldots, p_k\}$ el conjunto de itemsets determinado en la segunda etapa. En el primer paso de la tercera etapa, se forma un grupo con cada itemset $p \in S$ que tiene como description intencional al propio p. Sea $G = \{G_1, G_2, \ldots, G_k\}$ el conjunto de grupos formados. Posteriormente, en el segundo paso se busca el itemset $p' \in F$ que maximice la medida $MPQ(p', PS_{G_i})$ para algún grupo $G_i \in G$; donde PS_{G_i} denota al conjunto de itemsets que describe al grupo G_i.

En el conjunto de itemset de F a evaluar en el paso anterior solo se tienen en cuenta aquellos que cumplan: a) $p' \in F \setminus S$ y b) $|cov(p') \cap cov(PS_{G_i})| \leq \alpha \cdot min(|cov(p')|, |cov(PS_{G_i})|)$. Sea p_{best} y G_{best} el par itemset-grupo tal que $MPQ(p_{best}, PS_{G_{best}})$ es máximo. Si $MPQ(p_{best}, PS_{G_{best}}) > 0$ y G_{best} es único, entonces se adiciona el itemset p' a la descripción intencional del grupo G_{best} y se repite el proceso anterior, desde el segundo paso. En otro caso, si $MPQ(p_{best}, PS_{G_{best}}) = 0$, o hay más de un grupo con el cual p' alcanza el máximo, entonces el proceso se detiene.

Sea $G = \{G_1, G_2, \ldots, G_k\}$ el conjunto de grupos formados hasta la tercera etapa. A partir de G se construye un conjunto $PS = \{PS_1, PS_2, \ldots, PS_k\}$, tal que cada PS_i es el conjunto de itemsets frecuentes que describe al grupo $G_i \in G$. En la cuarta etapa se lleva a cabo un proceso que tiene como objetivo asignar el resto de los itemsets en $F \setminus \bigcup_{i=1}^{k} PS_i$ a la descripción intencional de algún grupo. Para esto, se recorre cada itemset $p' \in F \setminus \bigcup_{i=1}^{k} PS_i$ y se determina el grupo G_i tal que la medida $PCM(p', G_i)$. Esta medida estima la pertenencia de un itemset a un grupo, con base en la descripción intencional de dicho grupo. La fórmula de esta medida es la siguiente:

$$PCM(p', G_i) = \frac{|cov(p') \cap cov(PS_{G_i})|}{PQ1(PS_{G_i})}.$$

Una vez identificado el grupo G_i con el cual p' alcanza el máximo de esta medida, p' es adicionado al conjunto PS_i. Si más de un grupo alcanza el máximo, dicho itemset no se asigna a ningún conjunto de itemsets del conjunto PS. Por último, en la quinta etapa se recorre cada objeto $o_j \in O$ y se asigna dicho objeto a la descripción extensional del grupo $G_i \in G$ que maximice la medida $TCM(o_j, G_i)$. Si para algún objeto existe más de un grupo que maximiza esta medida, entonces dicho objeto no se asigna a ningún grupo. La medida $TCM(o_j, G_i)$ calcula la membresía de un objeto o_j a un grupo G_i, con base en el conjunto de itemsets $PS_i \in PS$ asociados a dicho grupo. La fórmula de esta medida es la siguiente:

$$TCM(o_j, G_i) = \sum_{p} \{vote(p) \mid p \in PS_i \wedge o_j \in cov(p)\},$$

donde la función $vote(p)$ se calcula como sigue:

$$vote(p) = \frac{PCM(p, G^1) - PCM(p, G^2)}{\sum_{i=1}^{k} PCM(p, G_i)} \cdot \left(\frac{|p_{max}|}{mgLen(p)}\right)^2,$$

donde G^1 y G^2 son el primer y segundo grupo de G que alcanzan los dos mayores valores de la medida $PCM(p, G)$.

Entre las limitaciones de este algoritmo está que solo procesa objetos descritos por atributos cualitativos. Si bien es cierto que los valores numéricos pueden ser discretizados, esta operación hace que se pierda la semántica de los datos y no garantiza que se alcancen buenos resultados de eficacia. Por otro lado, el algoritmo tiene una gran complejidad computacional, derivada sobre todo por la gran cantidad de cálculos que tiene que hacer. Aunque en [107] se brindan algunas optimizaciones para acelerar el procesamiento de CPC, estas optimizaciones hacen que el algoritmo consuma mucha memoria y por lo tanto, no sea útil en problemas que procesen una gran cantidad de objetos. Adicionalmente, CPC requiere ajustar los valores de cinco parámetro, incluido el umbral de soporte s_{min}. Si bien es cierto que los autores proponen valores para la mayoría de los parámetros, no lo dan para s_{min}. Por último, CPC no garantiza que la descripción intencional de cada grupo cubra a todos los objetos del grupo, ni que todos los objetos queden agrupados.

2.21.12. MFI Kmeans y SDHC

MFI Kmeans [109] y SDHC [110] son algoritmos que combinan el cálculo de itemsets frecuentes y la eficiencia del algoritmo Kmeans, para construir un conjunto de grupos disjuntos y sus descripciones, a partir de una colección de documentos. Ambos algoritmos asumen que cada documento está representado como un vector de términos que tienen asociado un peso que expresa la importancia del término para describir al documento; el peso se determina utilizando el esquema de pesado *tf-idf* [66].

Para construir un conjunto de grupos y sus descripciones, MFI Kmeans utiliza una estrategia compuesta de cinco pasos. En el primer paso se elimina de cada documento, aquellos términos cuya frecuencia en

la colección no supere un umbral α, previamente especificado. La frecuencia de un término en la colección es el número de documentos en los que el término está presente. En el segundo paso se utiliza el algoritmo MAFIA [86] para calcular el conjunto de itemset frecuentes maximales y se seleccionan los k itemset frecuentes maximales de mayor tamaño.

Definición 22 (conjunto de términos maximal). *Sea $f \subseteq T$ un itemset frecuente. Se dice que f es maximal si no existe otro itemset frecuente f' tal que $f \subset f'$.*

Sea $F = \{f_1, f_2, \ldots, f_k\}$ el conjunto de itemset frecuentes maximales seleccionados en el paso anterior. En el tercer paso, se forma un grupo con cada itemset en $f \in F$, que tiene al propio f como descripción intencional y al conjunto de documentos en $cov(f)$ como descripción extensional. Sea $G = \{G_1, G_2, \ldots, G_k\}$ el conjunto de grupos formados en el paso anterior. En el cuarto paso se calcula el centroide de cada grupo $G_i \in G$, denotado como C_{G_i}. Estos k centroides son procesados en el quinto paso por el algoritmo Kmeans, para hallar el agrupamiento final.

La principal limitación del algoritmo MFI Kmeans es que necesita ajustar valores de tres parámetros: α, el umbral s_{min} y k. Como se ha mencionado anteriormente, la asignación de valores para estos parámetros es compleja en problemas reales. Además, los parámetros s_{min} y k determinan la eficacia y eficiencia del algoritmo. Por otra parte, MFI Kmeans no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

Para formar un agrupamiento conceptual, el algoritmo SDHC emplea cuatro pasos. En el primer paso se utiliza el algoritmo parTFI [111], para determinar el conjunto F de itemsets *top-k frecuentes*. Un itemset f se considera top-k frecuente de longitud min_L, si el mismo es frecuente y no existe más de k-1 itemset, de longitud min_L, cuyo soporte sea mayor que el de f. En el segundo paso se crea un grupo con cada itemset $f_i \in F$ que tiene al propio f_i como descripción intencional y al conjunto de documentos en $cov(f_i)$ como descripción extensional. Sea $G = \{G_1, G_2, \ldots, G_k\}$ el conjunto de grupos formados en el paso anterior. En el tercer paso se determina el centroide de cada grupo G, denotado por C_G, como el promedio de todos los documentos presentes en el grupo G. De forma similar al algoritmo MFI Kmeans, los centroides y los documentos de la colección son procesados en el cuarto paso, por el algoritmo Kmeans, para formar el agrupamiento final.

Entre las limitaciones de SDHC está que necesita ajustar valores para varios parámetros: para el umbral de soporte, para el valor k utilizado para calcular los itemsets top-k frecuentes, para el parámetro min_L. Los valores de estos parámetros dependen mucho de la colección a agrupar. Por otra parte, SDHC no es capaz de procesar objetos descritos por atributos numéricos, mezclados, ni incompletos.

2.22. Maple

Maple [112] es un algoritmo que permite formar un agrupamiento, en el que se permite el traslape entre los grupos y éstos tienen asociado una descripción extensional e intensional. Este algoritmo asume que los objetos están descritos por un conjunto de atributos numéricos. Maple está basado en el concepto de γ-pCluster.

Sea $O = \{o_1, o_2, \ldots, o_n\}$ una colección de objetos descritos por un conjunto de atributos numéricos $A = \{a_1, a_2, \ldots, a_m\}$. Sea (R, T) un par de conjuntos tal que $R \subseteq O$ y $T \subseteq A$. El par (R, T) es un γ-pCluster si para cualquier par de objetos $o_i, o_j \in R$ y cualquier par de atributos $a_q, a_s \in T$, se cumple que $\| (o_i.a_q - o_j.a_q) - (o_i.a_s - o_j.a_s) \| \leq \gamma$; donde $\gamma \geq 0$ es un parámetro del algoritmo. El agrupamiento que construye Maple está formado por todos los γ-pCluster significantes y maximales.

Sean $C_1 = (R_1, T_1)$ y $C_2 = (R_2, T_2)$ dos γ-pCluster. C_1 es un sub-grupo propio de C_2 si $R_1 \subset R_2$ o $R_1 \subset R_2$. Un γ-pCluster C es maximal (en lo siguiente referido como γ-MPC) si no existe otro γ-pCluster

C' tal que c es un sub-grupo propio de C'. Un γ-pCluster $C_1 = (R_1, T_1)$ es significante dado dos umbrales min_{obj} y min_{atr}, si se cumple que $|R_1| \geq min_{obj}$ y $|T_1| \geq min_{atr}$.

Para construir el conjunto de todos los γ-pCluster significantes y maximales, se sigue una estrategia compuesta de cuatro etapas. En la primera etapa se calcula el conjunto de γ-MPC que contienen solo dos atributos (γ-MPC$_A$) o solo dos objetos (γ-MPC$_O$). Para calcular los γ-MPC$_A$ respecto a un par de atributos $a_1, a_2 \in A$, se ordenan los objetos $o_i \in O$ en orden creciente, de acuerdo a la diferencia que existe entre $o_i.a_1$ y $o_i.a_2$, Posteriormente, se calculan de esta lista ordenada, todos las sub-cadenas de objetos consecutivos $\{o_1, o_2, \ldots, o_q\}$ tal que $\| (o_i.a_1 - o_q.a_1) - (o_i.a_2 - o_q.a_2) \| \leq \gamma$. Sean $\{s_1, s_2, \ldots, s_k\}$ las sub-cadenas $s_i \subseteq O$ que cumplen la condición anterior. Cada par $(s_i, \{a_1, a_2\})$ forma un γ-MPC$_A$. El procedimiento anterior se ejecuta para cada par de atributos $a_i, a_j \in A$.

Una vez que se calculan todos los γ-MPC$_A$, se realiza un proceso de filtrado de objetos y atributos, con base en el lema 3.1 presentado en [112]. En el primer paso, se elimina del conjunto de atributos A todo aquel atributo que aparezca en menos de min_{atr} - 1, de los γ-MPC$_A$ calculados. Posteriormente, en el segundo paso se eliminan de O todos los objetos que aparecen en menos de $\frac{min_{atr} \cdot (min_{atr}-1)}{2}$, de los γ-MPC$_A$ calculados. De esta forma, se reduce el espacio de búsqueda de los γ-MPC$_O$. Para calcular los γ-MPC$_O$ asociados a cada par de objetos $o_i, o_j \in O$, se sigue una estrategia similar a la empleada para calcular los γ-MPC$_A$.

Cuando se determinan todos los γ-MPC$_A$ y los γ-MPC$_O$, en la segunda etapa se realiza un proceso de filtrado de los mismos. En el primer paso, se cuenta la aparición de cada objeto y cada atributo en los γ-MPC$_A$, eliminándose todo atributo y objeto que aparezca en menos de min_{atr} - 1 y $\frac{min_{atr} \cdot (min_{atr}-1)}{2}$, de los γ-MPC$_A$. En el segundo paso, se eliminan todos los γ-MPC$_O$ que tengan menos de min_{atr} atributos. Posteriormente, en el tercer paso se cuenta la aparición de cada objeto y cada atributo en los γ-MPC$_O$, eliminándose todo objeto y atributo que aparezca en menos de min_{obj} - 1 y $\frac{min_{obj} \cdot (min_{obj}-1)}{2}$, de los γ-MPC$_O$. En el cuarto paso, se eliminan todos los γ-MPC$_A$ que tengan menos de min_{obj} objetos. Los cuatro pasos anteriores se repiten hasta que no se elimine ningún otro γ-MPC$_A$ y γ-MPC$_O$.

En la tercera etapa, se calcula el ranking de cada atributo $a_i \in A$ y se construye una lista AL de todos los atributos, ordenados por su *ranking*. El ranking de un atributo se calcula como el número de objetos que están almacenados en la unión de todos los γ-MPC$_A$, en los que está incluido dicho atributo. En la cuarta etapa se recorre cada par de atributos $a_i, a_j \in AL$ y se construye el conjunto $Max - Fil$ de γ-pCluster maximales por fila que tienen al par a_i, a_j como atributos; este conjunto se denota como $Max - Fil^{\{a_i, a_j\}}$ y en este caso, coincide con los γ-MPC$_A$ que contienen al par a_i, a_j. Posteriormente, para cada γ-pCluster maximal por fila $(R, \{a_i, a_j\}) \in Max - Fil^{\{a_i, a_j\}}$, se invoca un método recursivo llamado *Search*.

Sea (R, A') el γ-pCluster maximal por fila que recibe el método Search como parámetro. En el primer paso del método Search, se construye la lista PD de posibles atributos con los que se puede extender el γ-pCluster (R, A'). Esta lista estará formada por todos los atributos $a' \in AL$ que estén contenidos en al menos $\frac{min_{obj} \cdot (min_{obj}-1)}{2}$ de los γ-MPC$_O$ que contengan a un par de objetos de R, y cuyo ranking sea menor que el del último atributo contenido en A'. Si $|PD \cup A'| < min_{atr}$, entonces termina el método y comienza el retroceso al llamado anterior a Search (i.e., *backtracking*). En otro caso, se recorre el conjunto de atributos de PD y, para cada atributo $a_i \in PD$, se busca el conjunto $Max - Fil^{A' \cup \{a_i\}}$ de γ-pCluster maximales por fila que tienen al conjunto $A' \cup \{a_i\}$ como atributos. Si $|A' \cup \{a_i\} \geq 3|$, entonces $Max - Fil^{A' \cup \{a_i\}}$ es el conjunto $\{(R'_{a_1,a_i}, A' \cup \{a_i\}), (R'_{a_2,a_i}, A' \cup \{a_i\}), \ldots, (R'_{a_{|A'|},a_i}, A' \cup \{a_i\})\}$; donde para $j = 1 \ldots |A'|, a_j \in A'$ y $R'_{a_j,a_i} = R \cap R_{a_j,a_i}$, siendo R_{a_j,a_i} el conjunto de objetos que tienen en común todos los γ-MPC$_A$ que tienen como par de atributos a a_j, a_i. Posteriormente, para cada elemento en $(R'_{a_1,a_i}, A' \cup \{a_i\}) \in Max - Fil^{A' \cup \{a_i\}}$ se llama recursivamente a Search con $(R'_{a_1,a_i}, A' \cup \{a_i\})$ como parámetro. Una vez que se terminan estos llamados recursivos y se termina de recorrer la lista PD, si el γ-pCluster recibido

como parámetro no es sub-cluster de ningún γ-MPC encontrado en algún llamado, entonces el γ-pCluster recibido como parámetro es un γ-MPC. Luego, se termina el método y comienza el retroceso al llamado anterior a Search (i.e., *backtracking*). Esta forma de construir los γ-MPC garantiza que todos los γ-MPC hallados sean significantes para los valores de umbrales min_{obj} y min_{atr}.

El conjunto de γ-MPC construidos por este método constituye el agrupamiento y en éste, cada γ-MPC (R,A) es un grupo que tiene como descripción extensional a los objetos de R y como descripción intencional a los atributos de A.

La principal limitación del algoritmo Maple es su alta complejidad computacional, producto del mismo procedimiento recursivo que emplea para formar los γ-MPC. Este aspecto hace a Maple poco útil en problemas en los que se trabaje con un gran número de objetos descrito por un alto número de atributos. Por otra parte, el algoritmo necesita ajustar los valores de los parámetros γ, min_{obj} y min_{atr}, los cuales dependen de la colección que se procesa e influyen en gran medida en la eficiencia y eficacia del algoritmo. Por último, Maple no es capaz de procesar objetos descritos por atributos cualitativos, mezclados, ni incompletos.

3. Discusión

En esta sección, se realiza una comparación cualitativa de los algoritmos descritos en la sección 2.1.2, atendiendo a un conjunto de características. Primeramente, en la subsección 3.1 se mencionan cuáles son los aspectos que se tendrán en cuenta en la comparación, por qué es interesante tenerlos en cuenta, así como los valores que se tendrán en cuenta para cada uno. Por último, en la subsección 3.2 se presenta la comparación entre los algoritmos y como resultado de ésta, se realizan algunas observaciones sobre los algoritmos.

3.1. Aspectos a medir en la comparación

Una de las características que siempre se tiene en cuenta durante el análisis de un algoritmo es su complejidad computacional, generalmente expresada por su desempeño en el peor de los casos. Mientras más baja sea la complejidad computacional mayor será el volumen de datos que podrá procesar el algoritmo. Años atrás, algoritmos con complejidades $O(n^2)$ eran considerados eficientes, de acuerdo a las características de las colecciones existentes. Hoy en día, con el aumento de las capacidades de almacenamiento y transmisión de datos, dichos algoritmos son poco eficientes. Con base en lo anterior, la complejidad computacional, referida en lo siguiente como "complejidad", será un factor a tener en cuenta en la comparación entre los algoritmos. Esta variable tomará valor *muy alta*, para aquellos algoritmos cuya complejidad sea mayor a $O(n^5)$, *alta* cuando la complejidad sea mayor a $O(n^3)$, *media* cuando la complejidad sea mayor a $O(n^2)$ y *baja* para aquellos algoritmos cuya complejidad sea menor a $O(n^2)$.

Otra característica interesante en los algoritmos es el tipo de estructuración que forman. Generalmente, la mayoría de los algoritmos de agrupamiento se han centrado en producir conjuntos de grupos, dejando al especialista la tarea de identificar las relaciones existentes entre dichos grupos. Como solución a esto, se han desarrollado algoritmos que detectan algunos tipos de relaciones y por ejemplo, forman jerarquías o retículos de grupos. Con base en lo anterior, el tipo de estructuración, en lo siguiente referido como "estructuración" será un factor a tener en cuenta en la comparación entre los algoritmos. Esta variable tomará los valores *conjunto*, *jerarquía* o *retículo* en dependencia de la estructura que forme el algoritmo.

Otro factor importante en un algoritmo es el número de parámetros de los cuales éste depende. Mientras mayor sea el número de parámetros del cual depende un algoritmo, más difícil será su aplicación en

un problema real. Incluso, como se comentó en la sección 2.1.2, hay parámetros que influyen considerablemente en la eficiencia y eficacia del algoritmo. Con base en lo anterior, el número de parámetros que requiera el algoritmo, en lo siguiente referido como "parámetros" será un factor a tener en cuenta en la comparación entre los algoritmos. Esta variable tomará valores enteros, de acuerdo a los parámetros de cada algoritmo.

Con el incremento en las capacidades de producir y almacenar información, así como con el desarrollo de diversos servicios sobre internet, han surgido nuevos requerimientos para los algoritmos de agrupamiento. Uno de estos requerimientos es la necesidad de procesar colecciones que pueden cambiar en el tiempo producto de adiciones, eliminaciones o modificaciones de la información contenida en la misma. En este sentido, es necesario comprender que los algoritmos que desechan los resultados obtenidos y vuelven a construir la estructuración desde cero, cuando la colección cambia, son muy ineficientes en contextos donde la información cambia con frecuencia. Con base en lo anterior, la capacidad de un algoritmo para procesar cambios, en lo siguiente referido como "tipo", será un factor a tener en cuenta en la comparación entre los algoritmos. Esta variable tomará los valores *estático*, *incremental* o *dinámico*, en dependencia si el algoritmo no permite procesar cambios, solo procesa adiciones o si es capaz de procesar adiciones y eliminaciones, respectivamente. Por lo general, las modificaciones se procesan como una eliminación seguida de una adición; por esta razón, no se tuvo en cuenta un valor para expresar la capacidad de procesar modificaciones de objetos.

En el mundo real, los objetos se describen tanto por atributos numéricos como cualitativos. Incluso, pueden haber contextos en los que, debido a una pérdida de información, no se cuente con el valor de todos los atributos de los objetos del universo de estudio; i.e., hayan atributos incompletos. Aunque bien es cierto que se han reportado estrategias para procesar de una forma alternativa los objetos descritos por rasgos mezclados e incompletos, la mayoría de los algoritmos procesan por separado estos atributos, discretizan los atributos numéricos a riesgo de perder la semántica del problema, estiman valores para los atributos incompletos de forma heurística a riesgo de introducir ruido en el problema o, simplemente, no tienen en cuenta estos atributos para el análisis. Con base en lo anterior, el tipo de atributos que puedan procesar los objetos, en lo siguiente referido como "atributos", será un factor a tener en cuenta en la comparación entre los algoritmos. Esta variable tomará los valores *numéricos*, *cualitativos*, *mezclados* o *mezc-inc*, en dependencia de los atributos que pueda procesar cada algoritmo.

Los grupos construidos por un algoritmo de agrupamiento pueden ser de tres tipos: disjuntos, solapados (o con traslape) y difusos. La gran mayoría de los algoritmos de agrupamiento reportados construyen grupos disjuntos. A pesar de que este acercamiento ha sido satisfactoriamente usado en la literatura, existen diferentes aplicaciones en las que resulta importante obtener grupos con traslape o difusos. Con base en lo anterior, el tipo de grupos que forman los algoritmos, en lo siguiente referido como "grupos", será un factor a tener en cuenta en la comparación entre los algoritmos. Esta variable tomará los valores *disjunto*, *difuso* o *traslape*, en dependencia de cómo sean los grupos que forme cada algoritmo.

Otro factor de interés es cómo los algoritmos representan a los objetos. Este factor en muchas ocasiones determina qué tipo de objetos puede procesar el algoritmo y por lo tanto, permite conocer el espectro de aplicación que tiene el mismo. Con base en lo anterior, la estructura que utilizan los algoritmos para representar a los objetos del universo de estudio, en lo siguiente referida como "representación", será un factor a tener en cuenta en la comparación entre los algoritmos. Esta variable tomará los valores *vectores*, *secuencias*, *grafos*, *árboles* o *pares a/v*.

Por último, un factor importante a tener en cuenta en el caso específico de los algoritmos de agrupamiento conceptual, es cuán difíciles resultan de interpretar los conceptos generados. En algunos casos, los algoritmos generan conceptos utilizando teoría de las probabilidades, grafos u otro lenguaje que puede resultar difícil de comprender para un usuario que no es especialista en el tema. En otras ocasiones, se

generan conceptos en un lenguaje sencillo, pero la longitud del concepto generado impide su comprensión. Con base en lo anterior, la facilidad de interpretación de los conceptos generados, en lo siguiente referida como "comprensión", será un factor a tener en cuenta en la comparación entre los algoritmos. Esta variable tomará los valores *difícil*, *promedio* o *fácil*, en dependencia de cuán difíciles de interpretar resulten los conceptos generados por un algoritmos, para un usuario final.

3.2. Comparación entre los algoritmos

En esta subsección se presenta una comparación entre los algoritmos descritos en la sección 2.1.2, atendiendo a las características seleccionadas en la subsección anterior. Es importante mencionar que en esta comparación no fueron incluidos aquellos algoritmos que fueran variantes de un algoritmo, en los cuales no se introdujera ningún cambio considerable respecto a la versión original. En este sentido, no se utilizarán en la comparación los algoritmos CLASSIT+, AGAPE, FTSC, FCDC, FCIHC y CFWMS. El primero y el segundo son variaciones simples de los algoritmos CLASSIT y DF, respectivamente. Por otra parte, FTSC es una extensión de FTC que solo modifica ligeramente la estrategia de FTC, eliminando uno de sus pasos. FTSC y FCDC son extensiones de FIHC que no introducen nuevos aportes y por último, CFWMS sigue la misma estrategia del algoritmo CFWS, pero en vez de utilizar secuencias de términos frecuentes, utiliza secuencias de conceptos frecuentes. Adicionalmente, de la familia de algoritmos CLUSTER solo se utilizarán en la comparación los algoritmos CLUSTER/2 y CLUSTER/3.

En la tabla 1, se presentan las características que tienen los algoritmos analizados, para cada uno de los aspectos de interés seleccionados en la subsección 3.1. Como se puede observar en esta tabla, en las columnas referidas a los factores de interés, hay varios valores que aparecen con un supra-índice. A continuación se explica qué quiere decir cada supra-índice utilizado en cada columna:

- Columna complejidad - En esta columna aparecen con el supra-índice "*"las complejidades de los algoritmos FTC, HFTC, FTSHC, FIHC, TDC, GPHC, MFI Kmeans, SDHC y el algoritmo de Krishna & Bhavani. La explicación a esto está en que aunque la complejidad de estos algoritmos por lo general no es elevada, dicha complejidad está sujeta al umbral de soporte y por lo tanto, puede crecer si este umbral toma valores muy bajos.
- Columna tipo - En esta columna aparecen con supra-índices las clasificaciones de los algoritmos COBBIT y HGA-COBWEB. En el caso de COBBIT, el supra-índice "3" expresa que aunque este algoritmo procesa eliminaciones y adiciones, no trabaja con toda la colección a la vez, sino con una ventana de tiempo de ésta. Por otro lado, el supra-índice "4" en el algoritmo HGA-COBWEB expresa que este algoritmo, a diferencia de otros algoritmos incrementales, es capaz de procesar múltiples adiciones de objetos.
- Columna atributos - En esta columna aparecen con supra-índices las clasificaciones de los algoritmos ITERATE, CKM y LINNEO+. En el caso de los algoritmos ITERATE y CKM, el supra-índice "5" expresa que aunque estos algoritmos pueden procesar objetos descritos por atributos numéricos y cualitativos, logran tal objetivo transformando los atributos cualitativos a numéricos o, los numéricos a cualitativos y no, procesando los objetos en su estado natural. En el caso de LINNEO+, el supra-índice "6" expresa que, aunque éste es capaz de procesar atributos incompletos, la forma de tratar con ellos es no teniéndolos en cuenta o asignarles un valor ficticio.
- Columna grupos - En esta columna aparecen con supra-índices las clasificaciones de los algoritmos WITT, EMO-CC, GPHC y FTC. En el caso de los tres primeros, el supra-índice "1" expresa que los algoritmos pueden dejar objetos sin agrupar. En el caso del algoritmo FTC, el supra-índice "7" expresa que este algoritmo puede modificarse para formar grupos con traslape y que dicha modificación está considerada por el autor del algoritmo.

- Columna parámetros - En esta columna aparecen con supra-índices las clasificaciones de los algoritmos GCC y el algoritmo de Henry *et al*. En el caso del primero, el supra-índice "8" expresa que este algoritmo requiere dos parámetros, siendo uno una lista de valores reales. En el caso del algoritmo de Henry *et al*, el supra-índice "9" expresa que este algoritmo propone además una variante, en la cual se estima automáticamente el parámetro que requiere el algoritmo.
- Columna comprensión - En esta columna aparecen con supra-índices las clasificaciones de los algoritmos EPAM, HDCC, LINNEO+, CPC y el algoritmo de Henry *et al*. En el caso de los tres primero, el supra-índice "2" expresa que los conceptos que generan para cada grupo pueden no cubrir a todos los objetos del grupo. En el caso del algoritmo de Henry *et al*, el supra-índice "10" expresa que este algoritmo propone una variante que produce descripciones difíciles de interpretar.

A partir de los datos incluidos en la tabla 1, se pueden hacer algunas observaciones interesantes. La primera es que la mayoría (32 de 52) de los algoritmos conceptuales son computacionalmente muy costosos; i.e., tienen complejidades altas o muy altas. La mayoría de los que tienen complejidades media son basados en patrones frecuentes y dependen del umbral de soporte mínimo, por lo que en la práctica pudieran ser costosos también. Aún más, no hay un algoritmo cuya complejidad computacional sea baja. Otra observación es que se ha trabajado más en el desarrollo de algoritmos jerárquicos que en el resto de los algoritmos, siendo los algoritmos que forman retículos los menos estudiados. Dentro de los algoritmos jerárquicos, los que brindan más facilidades de acuerdo a los factores evaluados son HFTC y FTSHC, ambos estáticos. Dentro de los algoritmos que forman retículos, el de mejor ubicación de acuerdo al resto de los parámetros evaluados es GALOIS. Dentro de los que forman un conjunto de grupos, los de mejor balance entre los factores medidos son FTC y el algoritmo de Henry *et al*.

Por otro lado, la mayoría de los algoritmos (36 de 52) son estáticos, lo cual expresa que solo una pequeña parte de los algoritmos conceptuales pueden procesar cambios en la colección. Dentro de los que pueden procesar cambios, los más eficientes de acuerdo a su complejidad son COBBIT, INC y LINNEO+; el primero es dinámico y los otros dos son incrementales. No obstante, los tres forman descripciones que resultan muy difíciles de interpretar. Lo anterior se traduce también en que, la mayoría de los algoritmos conceptuales que procesan cambios son muy costosos y por lo tanto, no están preparados para ser empleados en contextos que procesen gran cantidad de objetos y en los cuales, la colección cambie con mucha frecuencia. Otra parte de este problema es que, solo dos de los algoritmos que permiten procesar cambios forman descripciones fáciles de comprender (UNIMEN y RESEARCHER) y justamente, estos dos tienen una complejidad computacional alta y dependen de cinco parámetros.

La mayoría de los algoritmos solo pueden procesar objetos descritos por atributos cualitativos (32 de 52). Esto implica que, para poder procesar un problema real con ellos puede que se necesite transformar los atributos numéricos a cualitativos y eliminar los atributos incompletos, corriendo el riesgo de perder la semántica del problema en cuestión. La otra parte de este problema está en que la mayoría de los algoritmos capaces de procesar objetos descritos por atributos mezclados, son computacionalmente muy costosos: tienen complejidades alta o muy alta (10 de 15); siendo los mejores INC, CKM, CKMSF, CKMCF y LINNEO+. No obstante, a excepción de LINNEO+, los otros cuatro algoritmos dependen de tres o más parámetros, lo cual dificulta también su aplicación.

Como se esperaba, la mayoría de los algoritmos forman grupos disjuntos (38 de 52), siendo los algoritmos basados en análisis de conceptos formales o en patrones frecuentes los que en su mayoría permiten formar grupos con traslape. No obstante, la mayoría de los algoritmos que producen traslape no son eficientes desde el punto de vista computacional: tienen complejidades alta o muy alta (8 de 14); en este contexto los mejores son FTC, HFTC, FTSHC, TDC y GPHC. Resulta interesante también ver que no hay ningún algoritmo conceptual que forme grupos difusos. Por otro lado, la mayoría de los algoritmos procesan objetos que están descritos como vectores o pares atributo-valor (48 de 52). Dicho de otra forma, solo

Tabla 1. Comparación entre los algoritmos atendiendo a los aspectos seleccionados.

Algoritmos	complejidad	estructuración	tipo	atributos	grupos	representación	parámetros	comprensión
CLUSTER/2	muy alta	conjunto	estático	mezclados	disjunto	pares a/v	4	promedio
CLUSTER/3	muy alta	jerarquía	estático	mezclados	disjunto	pares a/v	4	promedio
WITT	media	conjunto	estático	cualitativos	disjunto[1]	pares a/v	3	difícil
EPAM	alta	jerarquía	incremental	cualitativos	disjunto	pares a/v	0	promedio[2]
UNIMEN	alta	jerarquía	incremental	mezclados	traslape	pares a/v	5	fácil
RESEARCHER	alta	jerarquía	incremental	mezclados	traslape	grafos	5	fácil
OPUS	muy alta	jerarquía	estático	cualitativos	disjunto	pares a/v	0	difícil
COBWEB	alta	jerarquía	incremental	cualitativos	disjunto	pares a/v	0	difícil
CLASSIT	alta	jerarquía	incremental	numéricos	disjunto	pares a/v	0	difícil
COBWEB/3	alta	jerarquía	incremental	mezclados	disjunto	pares a/v	0	difícil
DynamicWEB	muy alta	jerarquía	dinámico	cualitativos	disjunto	pares a/v	1	difícil
ARACHNE	muy alta	jerarquía	incremental	cualitativos	disjunto	pares a/v	0	difícil
COBBIT	media	jerarquía	dinámico[3]	cualitativos	disjunto	pares a/v	1	difícil
BRIDGER	muy alta	jerarquía	incremental	mezclados	disjunto	pares a/v	0	difícil
INC	media	jerarquía	incremental	mezclados	disjunto	pares a/v	3	difícil
HGA-COBWEB	muy alta	jerarquía	incremental[4]	mezclados	disjunto	pares a/v	3	difícil
ITERATE	muy alta	conjunto	estático	mezclados[5]	disjunto	pares a/v	0	difícil
CKM	media	conjunto	estático	mezclados[5]	disjunto	pares a/v	5	promedio
CKMSF	media	conjunto	estático	mezclados	disjunto	pares a/v	3	promedio
CKMCF	media	conjunto	estático	mezclados	disjunto	pares a/v	3	promedio
LC	muy alta	conjunto	estático	mezc-inc	disjunto	pares a/v	1	promedio
RGC	muy alta	conjunto	estático	mezc-inc	disjunto	pares a/v	1	promedio
EMO-CC	media	conjunto	estático	cualitativos	disjunto[1]	grafos	3	difícil
DF	alta	conjunto	estático	cualitativos	disjunto	graf./par.	3	promedio
MCC	alta	retículo	estático	numéricos	traslape	vectores	4	promedio
GALOIS	alta	retículo	incremental	cualitativos	traslape	pares a/v	0	promedio
Hoto & Stumme	alta	retículo	estático	cualitativos	traslape	pares a/v	2	promedio
HDCC	media	jerarquía	estático	numéricos	disjunto	vectores	0	promedio[2]
GCC	media	jerarquía	estático	cualitativos	disjunto	pares a/v	2[8]	difícil
LINNEO+	media	conjunto	incremental	mezc-inc[6]	disjunto	pares a/v	0	difícil[2]
COING	alta	retículo	estático	cualitativos	disjunto	grafos	0	difícil
KIDS	muy alta	retículo	estático	cualitativos	disjunto	grafos	0	difícil
M-DISC	alta	jerarquía	estático	numéricos	disjunto	pares a/v	1	difícil
LABYRINTH	muy alta	jerarquía	incremental	cualitativos	disjunto	vect./árb	0	difícil
Henry *et al*	media	conjunto	estático	cualitativos	disjunto	vectores	1[9]	fácil[1] 0
Aurora *et al*	muy alta	jerarquía	incremental	cualitativos	disjunto	vectores	1	difícil
FTC	media*	conjunto	estático	cualitativos	disjunto[7]	vectores	1	fácil
HFTC	media*	jerarquía	estático	cualitativos	traslape	vectores	1	fácil
FTSHC	media*	jerarquía	estático	cualitativos	traslape	vectores	1	fácil
FIHC	media*	jerarquía	estático	cualitativos	disjunto	vectores	1	fácil
TDC	media*	jerarquía	estático	cualitativos	traslape	vectores	2	fácil
GPHC	media*	jerarquía	estático	cualitativos	traslape[1]	vectores	3	fácil
Krishna & Bhavani	media*	conjunto	estático	cualitativos	disjunto	vectores	3	fácil
F^2IHC	alta	jerarquía	estático	cualitativos	disjunto	vectores	6	fácil
IDHC	media	jerarquía	estático	cualitativos	traslape	vectores	5	fácil
Kiran *et al*	alta	jerarquía	estático	cualitativos	traslape	vectores	4	difícil
CFWS	alta	conjunto	estático	cualitativos	traslape	secuencias	2	difícil
MC	media	conjunto	estático	cualitativos	disjunto	vectores	1	promedio
CPC	alta	conjunto	estático	cualitativos	disjunto	pares a/v	5	fácil[2]
MFI Kmeans	media*	conjunto	estático	cualitativos	disjunto	vectores	3	fácil
SDHC	media*	conjunto	estático	cualitativos	disjunto	vectores	3	fácil
Maple	alta	conjunto	estático	numéricos	traslape	pares a/v	3	difícil

seis algoritmos son capaces de procesar objetos descritos por estructuras complejas como grafos, árboles o secuencias.

Adicionalmente, casi la mitad de los algoritmos dependen de 2 o más parámetros (25 de 52), mientras que de la otra mitad, los algoritmos que dependen de uno o de ningún parámetro, la mayoría tienen complejidades alta o muy alta (18 de 27). Por otro lado, la mayoría de los algoritmos generan descripciones que son difíciles de interpretar, ya sea porque se requiere tener algún conocimiento en un área en específico o por que la longitud de la descripción es demasiado grande (37 de 52). La mayoría de los algoritmos que producen descripciones que resultan fáciles de interpretar están basados en patrones frecuentes (12 de 15), siendo los mejores, de acuerdo a los valores que tienen para el resto de los factores, los algoritmos FTC, HFTC, FTSHC, FIHC y el algoritmo de Henry *et al*. Dentro de los algoritmos basados en patrones, el patrón que más se utiliza es el itemset frecuente.

De todo lo anterior se puede aseverar que, de manera global, los algoritmos que más ventajas ofrecen para la construcción de agrupamientos conceptuales son los basados en patrones frecuentes, y específicamente, los algoritmos FTC, HFTC, FTSHC y FIHC. No obstante, todos estos son estáticos y por lo tanto, resultan poco útiles para las aplicaciones que procesan colecciones que cambian frecuentemente en el tiempo; la mayoría de las aplicaciones hoy en la actualidad.

4. Conclusiones

La posibilidad de formar descripciones de los grupos determinados, es una característica necesaria y deseable en los algoritmos de agrupamiento, sobre todo en aquellas aplicaciones en que los usuarios están más interesados en poder interpretar los resultados para poder sacar partida de ellos. En este reporte técnico se ha realizado un estudio del estado-del-arte en el área del agrupamiento conceptual, describiéndose los algoritmos más influyentes que se han reportado en la literatura y analizado críticamente las limitaciones de éstos.

Adicionalmente, se realizó una comparación cualitativa de los algoritmos descritos, de acuerdo a un conjunto de factores que denotan características deseables en los algoritmos de agrupamiento conceptual. Con base en esta comparación, se puede concluir que el desarrollo de algoritmos de agrupamiento conceptual, que sean capaces de procesar eficientemente colecciones de grandes volúmenes de objetos y permitan formar descripciones fáciles de interpretar, es todavía un problema abierto. Adicionalmente, se pueden hacer algunas otras conclusiones secundarias:

1. Todavía es insuficiente el desarrollo en el área del agrupamiento conceptual incremental y dinámico, sobre todo en esta última parte. Los algoritmos existentes hoy en día son poco útiles en la mayoría de las aplicaciones que procesan colecciones que cambian en el tiempo. Por lo tanto, una línea de investigación interesante y que se necesita potencializar, es la de desarrollar algoritmos conceptuales dinámicos que sean capaces de procesar eficientemente multiples adiciones y eliminaciones de objetos.
2. La mayoría de los algoritmos conceptuales no son aplicables en problemas reales en los cuales, por lo general, los objetos se describen por rasgos mezclados e incompletos. La mayoría de las soluciones para abordar este problema consiste en transformar las descripciones de los objetos o ignorando la parte de las descripciones que resulten problemáticas. Luego, otra línea de investigación que necesita ser potencializada es la de desarrollar algoritmos de agrupamiento conceptual, eficientes y eficaces, que sean capaces de procesar objetos descritos por atributos mezclados e incompletos.
3. El trabajo realizado en el área del agrupamiento conceptual con traslape es poco y nulo en el caso del agrupamiento conceptual difuso. Aunque en un principio todo el estudio de los métodos de agrupamiento se centró en formar particiones, hoy en día la mayoría de las aplicaciones requieren algoritmos

que permitan formar grupos con traslape. En este sentido, como se ha comentado anteriormente en este trabajo, contar también con la descripción de los grupos sería de gran utilidad en problemas reales. Por otra parte, la idea de poder formar grupos difusos permite reflejar mejor la realidad, en la cual, una persona o un objeto no tiene porqué pertenecer con el mismo grado a todos los grupos. Combinar la formación de grupos difusos, con la generación de descripciones para los grupos, es todo un desafío. Con base en los anterior, otra línea de investigación que requiere ser potencializada es la de desarrollar algoritmos de agrupamiento conceptual que permitan formar grupos con traslape. De igual forma, una nueva línea de investigación a abordar sería la del desarrollo de algoritmos de agrupamiento conceptual y difuso.

4. La mayoría de los algoritmos conceptuales que permiten detectar relaciones entre los grupos formados, se han centrado en la construcción de jerarquías. En tal sentido se debería explorar la búsqueda de otro tipo de relaciones, así como potencializar el desarrollo de algoritmos conceptuales que formen retículos.
5. Dadas las buenas características mostradas por los algoritmos basados en patrones frecuentes y conociendo que estos algoritmos son estáticos, una línea de investigación que resulta interesante es la de desarrollar algoritmos de agrupamiento conceptual, basados en patrones frecuentes, que permitan procesar cambios en las colecciones de objetos.
6. Dado la alta complejidad computacional de la mayoría de los algoritmos conceptuales y las grandes posibilidades de generación y almacenamiento de información que existe en la actualidad, una alternativa interesante para poder procesar colecciones de grandes volúmenes de forma eficiente, sería la de desarrollar algoritmos conceptuales que fueran paralelos.

Por último, dada las diferentes características que tienen los algoritmos reportados en este trabajo, una comparación cuantitativa entre todos ellos, a través de experimentos sobre colecciones, no sería del todo justa o no sería aplicable. Sin embargo, lo que sí se puede hacer y queda como trabajo futuro de este trabajo, es implementar estos algoritmos eficientemente y detectar, para aquellos algoritmos basados en los mismos supuestos y que permiten procesar el mismo tipo de colecciones, cuál o cuáles son los de mejor eficiencia y eficacia, de acuerdo a medidas de evaluación reportadas en la literatura.

Referencias bibliográficas

1. Pfitzner, D., Leibbrandt, R., Powers, D.: Characterization and Evaluation of Similarity Measures for Pairs of Clusterings. Knowledge and Information Systems **19**(3) (2009) 361–394
2. Hsiu-Hsia, L., San-Ging, S., Yueh-Huang, L., Shyr-Shen, Y.: Bone Age Cluster Assessment and Feature Clustering Analysis Based on Phalangeal Image Rough Segmentation. Pattern Recognition **45**(1) (2012) 322–332
3. Michel, V., Gramfort, A., Varoquaux, G., Eger, E., Keribin, C., Thirion, B.: A Supervised Clustering Approach for FMRI-based Inference of Brain States. Pattern Recognition **45**(6) (2012) 2041–2049
4. Li, Y., Shi, H., Jiao, L., Liu, R.: Quantum Evolutionary Clustering Algorithm Based on Watershed Applied to SAR Image Segmentation. Neurocomputing **87** (2012) 90–98
5. Ducournau, A., Bretto, A., Rital, S., Laget, B.: A Reductive Approach to Hypergraph Clustering: An Application to Image Segmentation. Pattern Recognition **45**(7) (2012) 2788–2803
6. Maraziotis, I.A.: A Semi-Supervised Fuzzy Clustering Algorithm Applied to Gene Expression Data. Pattern Recognition **45**(1) (2012) 637–648
7. Munir, M.U., Javed, M.Y., Khan, S.A.: A Hierarchical K-means Clustering Based Fingerprint Quality Classification. Neurocomputing **85** (2012) 62–67
8. Martínez-Trinidad, J.F., Guzman-Arenas, A.: The Logical Combinatorial Approach to Pattern Recognition An Overview Through Selected Works. Pattern Recogn. **34**(4) (2001) 1–11
9. Jain, A.K., Murty, M.N., Flynn, P.J.: Data Clustering: A Review. ACM Computing Surveys **31**(3) (1999) 264–323
10. Feigenbaum, E.A.: The Simulation of Verbal Learning Behavior. In: Proceedings of the Western Joint IRE-AIEE-ACM Computer Conference. (1961) 121–132

11. Michalski, R.S.: Knowledge Acquisition Through Conceptual Clustering: A Theoretical Framework and An Algorithm for Partitioning Data into Conjunctive Concepts. A Special Issue on Knowledge Acquisition and Induction, International Journal of Policy Analysis and Information Systems **4**(3) (1980) 219–244
12. Michalski, R.S.: Conceptual Clustering: A Theoretical Foundation and A Method for Partitioning Data into Conjunctive Concepts. In: Seminaries IRIA, Classification Automatique et Perception par Ordinateur, INRIA, France. (1979) 253–295
13. Michalski, R.S., Stepp, R.E.: Automated Construction of Classifications: Conceptual Clustering Versus Numerical Taxonomy. IEEE Trans. Pattern Anal. Mach. Intell. **5**(4) (1983) 396–410
14. Stepp, R.E., Michalski, R.S.: Conceptual Clustering: Inventing Goal Oriented Classifications of Structured Objects. In: Machine Learning: An Artificial Intelligence Approach, Vol II. (1986) 471–498
15. Seeman, W.D., Michalski, R.S.: The CLUSTER/3 System for Goal-Oriented Conceptual Clustering: Method and Preliminary Results. In: Proceedings of The Data Mining and Information Engineering 2006 Conference. (2006) 81–90
16. Michalski, R.S.: Pattern Recognition as Rule-Guided Inductive Inference. IEEE Trans. Pattern Anal. Mach. Intell. **2**(4) (1980) 349–4361
17. Michalski, R.S.: A Theory and Methodology of Inductive Learning. In: Machine Learning: An Artificial Intelligence Approach. (1983) 83–134
18. Hoff, W., Michalski, R.S., Stepp, R.E.: INDUCE/2: A Program for Learning Structural Descriptions from Examples. Technical Report UJUCDCS-F-83-904, Department of Computer Science, University of Illinois at Urbana-Champaing (1983)
19. Hanson, S.J., Bauer, M.: Conceptual Clustering, Categorization, and Polymorphy. Machine Learning **3**(4) (1989) 343–372
20. Sibson, R.: An Optimally Efficient Algorithm for the Single Link Cluster Method. The Computer Journal **16**(1) (1973) 30–34
21. Talmon, J.L., Fonteijn, H., Braspenning, P.J.: An Analysis of the WITT Algorithm. Machine Learning **11**(1) (1993) 91–104
22. Lebowitz, M.: Experiments With Incremental Concept Formation: UNIMEM. Machine Learning **2**(2) (1987) 103–138
23. Fisher, D.H.: Knowledge Acquisition Via Incremental Conceptual Clustering. Machine Learning **2**(2) (1987) 139–172
24. Lebowitz, M.: Concept Learning in A Rich Input Domain: Generalization-Based Memory. Machine Learning:An Artificial Intelligence Approach **2** (1986) 193–214
25. Nordhausen, B.: Conceptual Clustering Using Relational Information. In: Proceedings of the AAAI-86. (1986) 505–512
26. Jaccard, P.: The Distribution of Flora in the Alpine Zone. The New Phytologist **11**(2) (1912) 37–50
27. Gennari, J.H., Langley, P., Fisher, D.H.: Models of Incremental Concept Formation. Artificial Intelligence **40**(1-3) (1989) 11–61
28. McKusick, K.B., Thompson, K.: COBWEB/3: A Portable Implementation. Technical Report FIA-90-6-18-2, NASA Ames Research Center (1990)
29. Sahoo, N., Callan, J., Krishnan, R., Duncan, G., Padman, R.: Incremental Hierarchical Clustering of Text Documents. In: Proceedings of the 15th ACM International Conference on Information and Knowledge Management. (2006) 357–366
30. Scanlan, J., Hartnett, J., Williams, R.: DynamicWEB: Profile Correlation Using COBWEB. In: Proceedings of the Australian Conference on Artificial Intelligence 2006, LNAI 4304. (2006) 1059–1063
31. Scanlan, J., Hartnett, J., Williams, R.: DynamicWEB: Adapting to Concept Drift and Object Drift in COBWEB. In: Proceedings of the Australian Conference on Artificial Intelligence 2008, LNAI 5360. (2008) 454–460
32. McKusick, K.B., Langley, P.: Constraints on Tree Structure in Concept Formation. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence. (1991) 810–816
33. Kilander, F., Jansson, C.G.: COBBIT - A Control Procedure for COBWEB in the Presence of Concept Drift. In: Proceedings of the European Conference on Machine Learning. (1993) 244–261
34. Reich, Y., Fenves, S.J.: The Formation and Use of Abstract Concepts in Design. In: Concept Formation: Knowledge and Experience in Unsupervised Learning. (1991) 323–353
35. Hadzikadic, M., Yun, D.Y.Y.: Concept Formation by Incremental Conceptual Clustering. In: Proceedings of the International Joint Conference Artificial Intelligence. (1989) 831–836
36. Yoo, Y.P., Pettey, C.C., Yoo, S.: A Hybrid Conceptual Clustering System. In: Proceedings of the 1996 ACM 24th annual conference on Computer science. (1996) 105–114
37. Katz, S.M.: Distribution of Content Words and Phrases in Text and Language Modelling. Natural Language Engineering **2**(1) (1996) 15–59
38. Aho, A., Hopcroft, J., Ullman, J.: Data Structures and Algorithms. Addison-Wesley Publishing Company (1983)
39. Biswas, G., Weinberg, J., Li, C.: ITERATE: A Conceptual Clustering Method for Knowledge Discovery in Databases. In: Proceedings of Innovative Applications of Artificial Intelligence in the Oil and Gas Industry. (1994) 111–139
40. Biswas, G., Weinberg, J.B., Fisher, D.H.: ITERATE: A Conceptual Clustering Algorithm for Data Mining. IEEE Transactions on Systems, Man, and Cybernetics **28**(2) (1998) 100–111
41. Ralambondrainy, H.: A Conceptual Version of the K-means Algorithm. Pattern Recogn. Lett. **16**(11) (1995) 1147–1157

42. Ayaquica-Martínez, I.O., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A.: Conceptual K-Means Algorithm with Similarity Functions. In: Proceedings of the 9th Iberoamerican Congress on Pattern Recognition (CIARP 2005), LNCS 3773. (2005) 368–376
43. Ayaquica-Martínez, I.O., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A.: Conceptual K-Means Algorithm Based on Complex Features. In: Proceedings of the 10th Iberoamerican Congress on Pattern Recognition (CIARP 2006), LNCS 4225. (2006) 491–501
44. García-Serrano, J.R., Martínez-Trinidad, J.F.: Extension to K-means Algorithm for the Use of Similarity Functions. In: Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery Proceedings. Prague, Czech. Republic. (1999) 354–359
45. Martínez-Trinidad, J.F., Sánchez-Díaz, G.: LC: A Conceptual Clustering Algorithm. In: Proceedings of the Second International Workshop on Machine Learning and Data Mining in Pattern Recognition. (2001) 117–127
46. Pons-Porrata, A., Martínez-Trinidad, J.F., Ruiz-Shulcloper, J.: RGC: A New Conceptual Clustering Algorithm for Mixed Incomplete Data Sets. Mathematical and Computer Modelling **36**(11-13) (2002) 1375–1385
47. Sánchez, G., Lazo, M., Fuentes, O.: Genetic Algorithm to Compute Typical Testors With Minimum Weight. In: Proceedings of IV Iberoamerican Workshop on Pattern Recognition. Havana, Cuba. (1998) 207–213
48. Romero-Zaliz, R., Rubio-Escudero, C., Cordón, O., Harari, O., del Val, C., Zwir, I.: Mining Structural Databases: An Evolutionary Multi-Objetive Conceptual Clustering Methodology. In: Proceedings of the EvoWorkshops 2006, LNCS 3907. (2006) 159–171
49. Romero-Zaliz, R.C., Rubio-Escudero, C., Perren-Cobb, J., Herrera, F., Cordón, O., Zwir, I.: A Multiobjective Evolutionary Conceptual Clustering Methodology for Gene Annotation Within Structural Databases: A Case of Study on the Gene Ontology Database. IEEE Transactions on Evolutionary Computation **12**(6) (2008) 679–701
50. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2) (2002) 182–197
51. Velcin, J., Ganascia, J.G.: Default Clustering With Conceptual Structures. Journal on Data Semantics VIII **4380** (2007) 1–25
52. Velcin, J., Ganascia, J.G.: Topic Extraction With AGAPE. In: Proceedings of the 3rd International Conference on Advanced Data Mining and Applications. (2007) 377–388
53. Lippman, W.: Public Opinion. Ed. MacMillan, NYC (1922)
54. Putnam, H.: The Meaning of 'Meaning'. In: Mind, Language, and Reality, Cambridge University Press. (1975) 215–271
55. Rosch, E.: Cognitive Representations of Semantic Categories. Journal of Experimental Psychology **104**(3) (1975) 192–233
56. Ng, M.K., Wong, J.C.: Clustering Categorical Data Sets Using Tabu Search Techniques. Pattern Recognition **35**(12) (2002) 2783–2790
57. Ganascia, J.G., Velcin, J.: Clustering of Conceptual Graphs With Sparse Data. In: Proceedings of the 12th International Conference on Conceptual Structures. (2004) 156–169
58. Lee, J., Rajauria, P., Subodh, K.S.: A Model-Based Conceptual Clustering of Moving Objects in Video Surveillance. In: Proceedings of SPIE 6506, Multimedia Content Access: Algorithms and Systems. (2007)
59. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum Likelihood From Incomplete Data Via the EM Algorithm. Journal of the Royal Statistical Society. Series B (Methodological) **39**(1) (1977) 1–38
60. Carpineto, C., Romano, G.: GALOIS : An Order-Theoretic Approach to Conceptual Clustering. In: Proceedings of the 10th International Conference on Machine Learning. (1993) 33–40
61. Carpineto, C., Romano, G.: A Lattice Conceptual Clustering System and Its Application to Browsing Retrieval. Machine Learning **24**(2) (1996) 95–122
62. Pernelle, N., Ventos, V., Soldano, H.: ZooM: Alpha Galois Lattices for Conceptual Clustering. In: Proceedings of the Managing Specialization/Generalization Hierarchies MASPEGHI Workshop. (2003)
63. Ventos, V., Soldano, H., Lamadon, T.: Alpha Galois Lattices. In: Proceedings of the Fourth IEEE International Conference on Data Mining, (ICDM04). (2004) 555–558
64. Hotho, A., Stumme, G.: Conceptual Clustering of Text Clusters. In: Proceedings of the FGML Workshop. Vol. 37. (2002) 37–45
65. Zhao, Y., Karypis, G.: Evaluation of Hierarchical Clustering Algorithms for Document Datasets. In: Proceedings of the International Conference on Information and Knowledge Management. (2002) 515–524
66. Greengrass, E.: Information Retrieval: A Survey. Technical Report TR-R52-008-001 (2001)
67. Berry, M.: Survey of Text Mining, Clustering, Classification and Retrieval. Springer-Verlag (2004)
68. Funes, A., Ferri, C., Hernández-Orallo, J., Ramírez-Quintana, M.J.: Hierarchical Distance-Based Conceptual Clustering. In: Proceedings of ECML PKDD 2008, LNAI 5212. (2008) 349–364
69. Funes, A., Ferri, C., Hernández-Orallo, J., Ramírez-Quintana, M.J.: An Instantiation of Hierarchical Distance-Based Conceptual Clustering for Propositional Learning. In: Proceedings of PAKDD 2009, LNAI 5476. (2009) 637–646
70. Talavera, L., Béjar, J.: Generality-Based Conceptual Clustering With Probabilistic Concepts. IEEE Transactions on Pattern Analysis and Machine Intelligence **23**(2) (2001) 196–206

71. Béjar, J., Cortés, U.: LINNEO+: Herramienta Para la Adquisición de Conocimiento y Generación de Reglas de Clasificación en Dominios Poco Estructurados. In: Proceedings of the III Congreso Iberoamericano de Inteligencia Artificial (IBERAMIA 92), La Habana, Cuba. (1992) 471–781
72. Béjar, J., Cortés, U., Poch, M.: LINNEO: A Classification Methodology for Ill-Structured Domains. Technical Report LSI-93-22-R, Departament de Llenguatges i Sistemes Informatics (UPC) (1993)
73. Hill, D.R.: A Vector Clustering Technique. In Samuelson, ed.: Mechanized Information Storage, Retrieval and Dissemination. (1968) 501–508
74. Bournaud, I., Ganascia, J.G.: Accounting for Domain Knowledge in the Construction of a Generalization Space. In: Proceedings of the Third International Conference on Conceptual Structures (ICCS97). (1997) 446–459
75. Bournaud, I., Courtine, M., Zucker, J.D.: Abstractions for Knowledge Organization of Relational Descriptions. In: Proceedings of the 4th International Symposium on Abstraction, Reformulation and Approximation (SARA2000). (2000) 87–106
76. Bournaud, I., Courtine, M., Zucker, J.D.: KIDS: An Iterative Algorithm to Organize Relational Knowledge. In: Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2000). (2000) 217–232
77. Chein, M., Mugnier, M.L.: Conceptual Graphs: Fundamental Notions. Revue d'Intelligence Artificielle **6**(4) (1992) 365–406
78. Chu, W.W., Chiang, K., Hsu, C., Yau, H.: An Error-Based Conceptual Clustering Method for Providing Approximate Query Answers. Communications of the ACM **39**(12) (1996)
79. Langley, P., Thompson, K., Iba, W., Gennari, J.H., Allen, J.A.: An Integrated Cognitive Architecture for Autonomous Agents. Technical Report ARI Research Note 90-48, University of California (1990)
80. Thompson, K., Langley, P.: Concept Formation in Structured Domains. In: Concept Formation: Knowledge and Experience in Unsupervised Learning. Morgan Kaufmann (1991) 127–161
81. Anaya-Sánchez, H., Pons-Porrata, A., Berlanga-Llavori, R.: A New Document Clustering Algorithm for Topic Discovering and Labeling. In: Proceedings of the 12th Iberoamerican Congress on Pattern Recognition (CIARP 2008), LNCS 5197. (2008) 161–168
82. Anaya-Sánchez, H., Pons-Porrata, A., Berlanga-Llavori, R.: A Document Clustering Algorithm for Discovering and Describing Topics. Pattern Recognition Letters **31**(6) (2010) 502–510
83. Pons-Porrata, A., Berlanga-Llavori, R., Ruiz-Shulcloper, J.: Topic Discovery Based on Text Mining Techniques. Information Processing and Management **43**(3) (2007) 752–768
84. Pons-Porrata, A., Berlanga-Llavori, R., Ruiz-Shulcloper, J.: On-line Event and Topic Detection by Using the Compact Sets Clustering. Journal of Intelligent and Fuzzy Systems **12**(3-4) (2002) 185–194
85. Beil, F., Ester, M., Xu, X.: Frequent Term-Based Text Clustering. In: Proceedings of the FGML workshop. Vol. 37. (2002) 436–442
86. Agrawal, R., Imielinski, T., Swami, A.N.: Mining Association Rules Between Sets of Items in Large Databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD93). (1993) 207–216
87. Shi, Z., Ester, M.: Performance Improvement for Frequent Term-based Text Clustering Algorithm. Technical report, Simon Fraser University (2003)
88. Liu, X., he, P.: A Study on Text Clustering Algorithms Based on Frequent Term Sets. In: Proceedings of Advanced Data Mining and Applications, LNCS 3584. (2005) 347–354
89. Wang, H., Liu, X.: Study on Frequent Term Set-Based Hierarchical Clustering Algorithm. In: Proceedings of the Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2011). (2011) 1182–1186
90. Fung, B.C.M.: Hierarchical Document Clustering Using Frequent Itemsets. Master's thesis, Simon Fraser University (1999)
91. Fung, B., Wangand, K., Ester, M.: Hierarchical Document Clustering Using Frequent Itemsets. In: Proceedings of the SIAM International Conference on Data Mining 2003. (2003)
92. Baghel, R., Dhir, R.: A Frequent Concepts Based Document Clustering Algorithm. International Journal of Computer Applications **4**(5) (2010) 6–12
93. Kryszkiewicz, M., Skonieczny, L.: Hierarchical Document Clustering Using Frequent Closed Sets. Advances in Soft Computing **35** (2006) 489–498
94. Yu, H., Searsmith, D., Li, X., Han, J.: Scalable Construction of Topic Directory With Nonparametric Closed Termset Mining. In: Proceedings of the Fourth IEEE International Conference on Data Mining. (2004) 563–566
95. Wang, J., Han, J., Pei, J.: CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In: Proceedings of the Int. Conf. Knowledge Discovery and Data Mining (KDD 2003). (2003) 236–245
96. Malik, H.H., Kender, J.R.: High Quality, Efficient Hierarchical Document Clustering Using Closed Interesting Itemsets. In: Proceedings of the Sixth International Conference on Data Mining. (2006) 991–996
97. Malik, H.H.: Efficient Algorithms for Clustering and Classifying High Dimensional Text and Discretized Data Using Interesting Patterns. PhD thesis, Graduate School of Arts and Sciences, Columbia University (2008)

98. Zhao, Y., Karypis, G., Fayyad, U.: Hierarchical Clustering Algorithms for Document Datasets. Data Mining and Knowledge Discovery **10**(2) (2005) 141–168
99. Krishna, S.M., Bhavani, S.D.: Performance Evaluation of An Efficient Frequent Item Sets-Based Text Clustering Approach. Global Journal of Computer Science and Technology **10**(11) (2010) 60–68
100. Chun-Ling, C., Tseng, F.S., Liang, T.: Mining Fuzzy Frequent Itemsets for Hierarchical Document Clustering. Information Processing and Management **46**(2) (2010) 193–211
101. Malik, H.H., Kender, J.R.: Instance Driven Hierarchical Clustering of Document Collections. In: Proceedings of Local Patterns to Global Models Workshop, European Conference on Machine Learning and Practice of Knowledge Discovery in Databases. (2008)
102. Malik, H.H., Kender, J.R., Fradkin, D., Moerchen, F.: Hierarchical Document Clustering Using Local Patterns. Data Mining and Knowledge Discovery **21**(1) (2010) 153–185
103. Kiran, G.V.R., Shankar, R., Pudi, V.: Frequent Itemset Based Hierarchical Document Clustering Using Wikipedia As External Knowledge. In: Proceedings of the 14th International Conference on Knowledge-based and Intelligent Information and Engineering Systems: Part II. (2010) 11–20
104. Pudi, V., Haritsa, J.R.: Generalized Closed Itemsets for Association Rule Mining. In: Proceedings of the 19th International Conference on Data Engineering ICDE2003. (2003) 714–716
105. Li, Y., Chung, S.M., Holt, J.D.: Text Document Clustering Based on Frequent Word Meaning Sequences. Data Knowl. Eng. **64**(1) (2008) 381–404
106. Zhang, W., Yoshida, T., Tang, X., Wanga, Q.: Text Clustering Using Frequent Itemsets. Knowledge-Based Systems **23**(5) (2010) 379–388
107. Fore, N., Dong, G.: CPC: A Contrast Pattern Based Clustering Algorithm. In: Contrast Data Mining: Concepts, Algorithms and Applications. Chapman and Hall/CRC (2012) 197–216
108. Han, J., Pei, J., Yin, Y., Mao, R.: Mining Frequent Patterns Without Candidate Generation: A Frequent-Ppattern Tree Approach. Data Mining and Knowledge Discovery **8**(1) (2004) 53–87
109. Zhuang, L., Dai, H.: A Maximal Frequent Itemset Approach for Web Document Clustering. In: Proceedings of the Fourth International Conference on Computer and Information Technology. (2004) 970–977
110. Wang, L., Tian, L., Jia, Y., Han, W.: A Hybrid Algorithm for Web Document Clustering Based on Frequent Term Sets and K-Means. In: Advances in Web and Network Technologies, and Information Management. (2007) 198–203
111. Yongheng, W., Yan, J., Shuqiang, Y.: Parallel Mining of Top-K Frequent Items in Very Large Text Database. In: Proceedings of the 6th International Conference on Advances in Web-Age Information Management. (2005) 706–712
112. Pei, J., Zhang, X., Cho, M., Wang, H., Yu, P.S.: MaPle: A Fast Algorithm for Maximal Pattern-Based Clustering. In: Proceedings of the Third IEEE International Conference on Data Mining, ICDM 2003. (2003) 259–266

RT_024, abril 2014
Aprobado por el Consejo Científico CENATAV

Editor: Lic. Lucía González Bayona
Diseño de Portada: Di. Alejandro Pérez Abraham
RNPS No. 2143
ISSN 2072-6260
Indicaciones para los Autores:
Seguir la plantilla que aparece en www.cenatav.co.cu
C E N A T A V
7ma. No. 21812 e/218 y 222, Rpto. Siboney, Playa;
La Habana. Cuba. C.P. 12200
Impreso en Cuba