Optimizaciones al Algoritmo de Agrupamiento Compacto Jerárquico Dinámico

Optimizations for the Dynamic Compact Hierarchical Clustering Algorithm

Yenisleidi Lora Domínguez [1*], Fernando Artigas Fuentes [1], Adrian Fonseca Bruzón [2], Reynier Ortega Bueno [2]

[1] Departamento de Computación, FCMC, Universidad de Oriente, Santiago de Cuba, Cuba, {yeni,artigas}@csd.uo.edu.cu
[2] CERPAMID, Santiago de Cuba, Cuba, {adrian, reynier.ortega}@cerpamid.co.cu

[*] Autor para correspondencia: yeni@csd.uo.edu.cu

Resumen
En este artículo proponemos dos optimizaciones a un algoritmo de agrupamiento compacto jerárquico dinámico, este es aplicado a un conjunto de documentos con el objetivo de agruparlos por temáticas. La primera optimización que proponemos es una estructura compacta basada en grafos que posibilita un uso eficiente de la memoria principal; y la segunda una heurística de poda, que permite que los documentos se procesen en un tiempo menor que el usado por una implementación anterior. Los resultados obtenidos en los experimentos demuestran que con estas optimizaciones se logra agrupar un mayor número de documentos en menor tiempo.

Palabras clave: agrupamiento jerárquico dinámico, colección de documentos, estructura de dato eficiente, optimización.

Abstract
In this paper we propose two optimizations for the compact dynamic hierarchical clustering algorithm. This algorithm is applied to a set of documents in order to discover the structure of topics that they describe. The first one is a condense graph data structure that enables efficient use of main memory; and the second a prune graph heuristic, which allows documents to be processed in a shorter time than those used by a previous implementation. The results obtained in the experiments demonstrate that these optimizations allow processing more objects in less time.

Keywords: *document collection, dynamic hierarchical clustering, efficient data structure, optimization.*

Introducción

El adelanto tecnológico actual propicia que se genere continuamente y a gran velocidad una enorme cantidad de información. Por lo que cada vez se hace más necesaria la existencia de herramientas que permitan procesar y extraer la información más relevante de interés para los usuarios. Se han propuesto numerosos métodos para darle solución a este problema, entre ellos se encuentra el agrupamiento de documentos (Ruiz *et al,* 1999; Gil, 2005), que permite estructurar la información en grupos de forma tal que en un grupo se encuentren los documentos más parecidos entre sí y diferentes a los pertenecientes a otros grupos.

En la actualidad la información cambia con el transcurso del tiempo, ya sea porque se actualiza, deja de ser de interés o surge nueva información. Por eso generalmente se requieren algoritmos dinámicos, que actualicen el agrupamiento a medida que cambia el conjunto de datos, que sean capaces de agregar nuevos objetos al agrupamiento o eliminar aquellos que ya no existen, sin necesidad de empezar desde el principio cada vez que ocurra un cambio. Otro aspecto que resulta de interés en varias aplicaciones es poder proporcionar vistas de los datos a diferentes niveles de abstracción, lo que facilita la visualización de los mismos, para ello se ha trabajado en el desarrollo de algoritmos jerárquicos.

Cuando los datos a agrupar son documentos el problema es aún más complejo. Los documentos imponen un reto adicional a los algoritmos de agrupamiento, pues estos se caracterizan por su elevada dimensionalidad, debido a que cada palabra distinta que aparece en una colección es considerada una dimensión. Aún cuando esta contenga pocos documentos la cantidad de palabras diferentes es elevada. Esta característica incrementa el costo de la función utilizada para comparar documentos.

En la literatura se encuentran reportados varios algoritmos de agrupamientos, entre ellos se encuentra el Algoritmo de Agrupamiento Compacto Jerárquico Dinámico (ACJD) (Gil *et al*., 2010) basado en un marco general (Gil *et al*., 2006). Este comienza considerando que cada grupo está formado por un solo objeto, se parte de n grupos unitarios y se construye un grafo de $ß_0$-semejanza. En este grafo los vértices se corresponden con los grupos y existirá una arista de un grupo a otro si su semejanza, calculada mediante una medida de semejanza entre grupos, supera un umbral de semejanza mínima permitida $ß_0$. Luego, se construye un subgrafo de máxima $ß_0$-semejanza a partir del grafo de $ß_0$-semejanza, en él existirá un arco de un grupo σi a un grupo σj si se cumple que σj es el grupo más β_0-semejante a σi. A este subgrafo se le aplica un algoritmo de cubrimiento para obtener los grupos del siguiente nivel de la jerarquía. Este proceso se repite hasta que el grafo de $ß_0$-semejanza sea completamente inconexo.

Aunque con este algoritmo se obtienen buenos resultados, cuando es aplicado en colecciones con una gran cantidad de objetos, tiene un elevado consumo de memoria, debido a que almacena varias estructuras de datos, esto provoca

que se reduzca la cantidad máxima posible de objetos a agrupar. Es por ello que nuestra contribución se encuentra enmarcada en proponer una estructura compacta que mejore la eficiencia del método así como una heurística de poda de los grafos con la finalidad de reducir la información.

Materiales y métodos

En el algoritmo ACJD los grafos de $ß_0$-semejanza, máxima $ß_0$-semejanza, el cubrimiento y la jerarquía, son representados en estructuras de datos independientes, lo que significa que hay información redundante (se duplica información de los vértices y de las aristas). Debido a esta deficiencia proponemos una estructura de datos compacta capaz de representar simultáneamente todas las estructuras anteriores.

Además, el grafo de $ß_0$-semejanza se caracteriza por ser un grafo denso lo que provoca un alto consumo de memoria. Es por ello que se propone una heurística de poda de este grafo con el objetivo de almacenar solo aquellas aristas que tengan una mayor posibilidad de ser usadas en el agrupamiento.

A. Estructura de datos compacta multi-grafo

La estructura propuesta contiene varios grafos, uno por cada nivel de la jerarquía, en donde la información de cada objeto o grupo aparece una sola vez. En la Figura 1 se muestra un esquema general de la misma. Aunque el algoritmo no requiere que se obtenga una estructura en forma de árbol, por comodidad y para facilitar la navegabilidad por la jerarquía se adicionó una raíz ficticia, denotada por R.

La información almacenada en cada nodo es la siguiente:

- Un identificador. El cual debe ser generado de forma incremental cada vez que se adiciona un nuevo nodo.
- Un objeto asociado a ese nodo. El cual contiene la representación del objeto.
- Un apuntador al nodo que le sigue en el nivel. En caso de ser el último este valor es Nulo.
- Un apuntador a cada uno de sus hijos en el nivel inmediato inferior.
- Un apuntador al padre que se encuentra en el nivel inmediato superior.
- Las aristas del grafo de $ß_0$-semejanza que se forma con los nodos de su nivel.

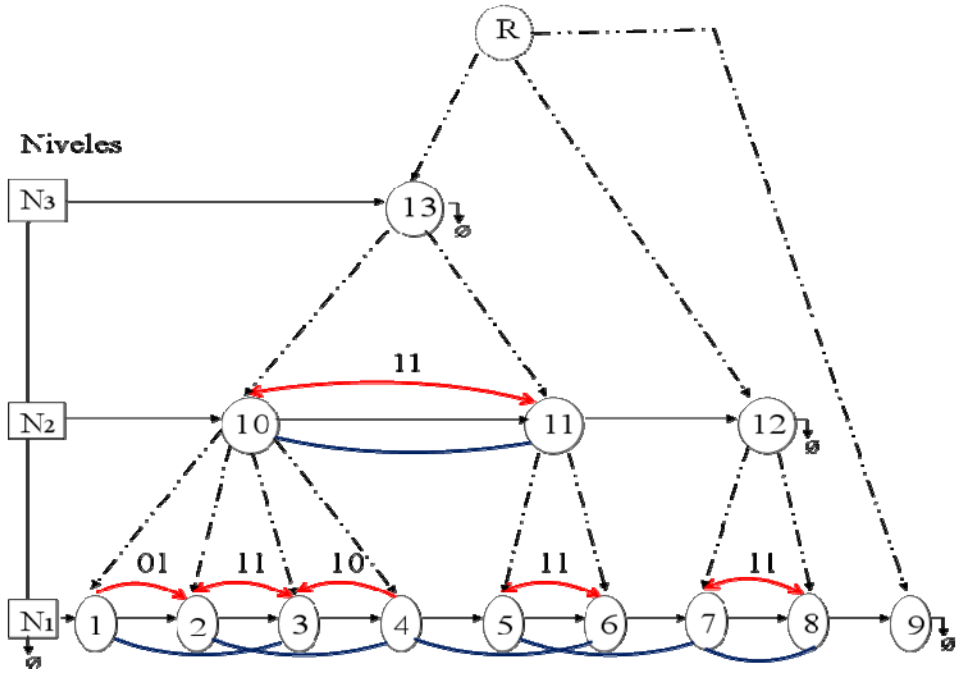

Figura 1. Estructura de datos compacta

La estructura cuenta además con una lista enlazada que permite acceder a cada uno de los niveles que forman parte de la estructura. En la Figura 1 esta lista enlazada está etiquetada con la palabra “Niveles”.

Cada arista en el grafo, que va desde el nodo 1 al nodo 2, contiene (se asume que el identificador del nodo 1 es menor que el de 2):

- Valor de semejanza entre los objetos de 1 y 2.
- Una etiqueta I que identifica la orientación de los arcos en el multi-grafo:

I = 00: Esta arista solo forma parte del grafo de $ß_0$-semejanzas, no hay un arco que una a los nodos 1 y 2 en el grafo de máxima $ß_0$-semejanzas.

I = 01: Existe un arco en el grafo de máxima $ß_0$-semejanza que va del nodo 1 al nodo 2.

I = 10: Existe un arco en el grafo de máxima $ß_0$-semejanza que va del nodo 2 al nodo 1.

I = 11: Este valor indica que existen ambos arcos entre 1 y 2. Uno va desde 1 hasta 2 y el otro de 2 hasta 1.

En la Figura 1, para facilitar la compresión, las aristas que solamente forman parte del grafo de $ß_0$-semejanza se encuentran dibujadas en color Azul. En color Rojo y con las saetas en correspondencia con las etiquetas I asociadas están representados los arcos que forman el grafo de máxima $ß_0$-semejanza.

Un aspecto importante a tener en cuenta, es que los nodos aislados de un determinado nivel no tienen padre en el nivel inmediato superior. En tal caso pasan a ser hijos de la raíz. Esto se hace para no tener que repetir información en los niveles superiores. Este es el caso del nodo 9 en el nivel N1 y del nodo 12 en el nivel N2 en el grafo de la Figura 1. Con esta estructura compacta se solapan los grafos de $ß_0$-semejanza y máxima $ß_0$-semejanza en uno a través de las etiquetas de las aristas, y la información del cubrimiento y la jerarquía a través de la relación padre-hijo.

B. Heurística de poda

Luego de realizar un análisis del consumo de memoria de esta propuesta, y teniendo en cuenta que implementaciones anteriores reducían este consumo mediante la estrategia de no almacenar el grafo de $ß_0$-semejanza, con la necesidad de calcular constantemente el mismo, se diseñó una estrategia intermedia para garantizar un equilibrio entre el consumo de memoria y el tiempo de ejecución.

Esta estrategia consiste en establecer un parámetro, denotado como α, para podar el grafo de $ß_0$-semejanza, manteniendo en este solamente aquellas aristas con mayor probabilidad de ser seleccionadas para formar parte del grafo de máxima $ß_0$-semejanza.

Concretamente se seleccionan aquellas aristas cuyo valor de semejanza entre los objetos que enlaza sea mayor que el parámetro $ß_0$ y además estén en el rango del máximo valor de semejanza menos el valor de α (ver Figura 2). Con el uso de un valor de α pequeño se ahorra en memoria pero se necesita más tiempo para actualizar el multi-grafo. En caso contrario usando un valor de α mayor, se gana en tiempo pero se necesita más memoria, ya que se almacena un mayor número de aristas del multi-grafo.

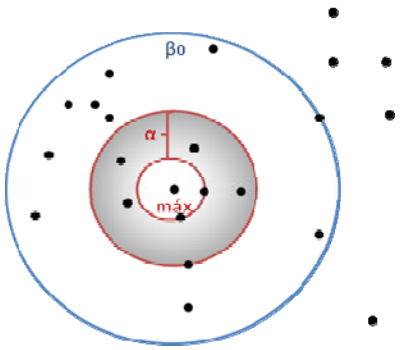

Figura 2. Poda del grafo de $ß_0$-semejanza

Este valor de α debe estar en el rango de [0,1], donde un valor de 1 implica almacenar todas las aristas del multi-grafo y un valor de 0 significa almacenar solamente las aristas que pertenecen al grafo de máxima $ß_0$-semejanza.

C. Algoritmo de agrupamiento compacto jerárquico dinámico

En esta sección mostraremos el funcionamiento del algoritmo ACJD adaptado a las optimizaciones propuestas. Los pasos básicos del método se muestran en el Algoritmo 1.

Algoritmo 1. ACJD

1. Dados los objetos a adicionar y/o a eliminar.
2. Se crean nuevos nodos en el multi-grafo con los objetos a adicionar.
3. nivel = 0.
4. Actualizar en el multi-grafo, el grafo de β_0–semejanza (*Gßnivel*) y el subgrafo de máxima β_0–semejanza (*GMnivel*).
5. Mientras *Gßnivel* no sea completamente inconexo:
   a. Actualizar el cubrimiento asociado al *GMnivel* (*Cnivel*).
   b. nivel = nivel+1.
   c. Actualizar en el multi-grafo *Gßnivel* y *GMnivel.*
6. Si existen niveles mayores que el último nivel alcanzado, eliminarlos y adicionar los vértices del último nivel a la raíz ficticia.

Los pasos básicos del proceso de actualización dinámica del grafo de β_0-semejanza y el subgrafo se muestran en Algoritmo 2.

Algoritmo 2. Actualización de *Gßnivel* y *GMnivel*.

1. Sea *N* el conjunto de vértices a agregar y *R* el conjunto de vértices a eliminar.
2. Sea *M* conjunto de vértices cuyos vértices más β_0-semejantes pertenecen a *R*, *RC* lista de componentes conexas y *NE* conjunto de aristas.
3. Eliminar los vértices de *R*, insertar en *RC* todos los vértices que tienen como hijo a un vértice en *R*.
4. Se adicionan los nuevos vértices como hijos de la raíz ficticia.
5. Calcular los valores de β_0-semejanza entre los vértices de *N* y el resto de los vértices del nivel y adicionar las aristas que superen el umbral β_0 y estén en el rango del máximo valor de semejanza menos α (Aristas con atributo I=00).
6. Obtener los vértices más β_0-semejantes a los vértices de *M* y *N,* actualizar las aristas en el multi-grafo (pertenecen a *GMnivel*), y almacenarlas en la lista *NE*.
7. Buscar los vértices para los cuales un elemento de *N* es su más β_0-semejante y luego actualizar las aristas (pertenecen a *GMnivel*) y adicionarlas a la lista *NE*.
8. Adicionar a *RC* las componentes a las que en el paso anterior se les eliminó alguna arista.
9. Retornar el conjunto de componentes *RC* y el conjunto de aristas adicionadas *NE*.

Cada vez que se elimina una arista del subgrafo, los objetos del grupo (conjunto compacto) al que pertenecen los vértices de esta arista pueden perder la propiedad de conexidad. Por tanto, este grupo debe ser reconstruido. Por el contrario, cada vez que se agrega una arista al subgrafo, los grupos de sus vértices se unen si son diferentes. La actualización de las componentes conexas provoca que aparezcan nuevos grupos y que se eliminen otros en el nivel superior (ver Algoritmo 3).

Algoritmo 3. Actualización de Cnivel

1. Sea *N* el conjunto de vértices adicionados, *RC* conjunto de las componentes conexas y *NE* conjunto de aristas adicionadas.
2. Sea *Q* una lista que contendrá todos los vértices que son necesarios procesar para la reconstrucción de las componentes conexas.
3. Adicionar a *Q* todos los vértices que tienen como padre a algún vértice de *RC* y los vértices de *N*.
4. Construir las componentes conexas existentes entre los vértices de *Q*.
5. Para cada arista en *NE*, unir las componentes conexas a los que pertenecen sus vértices si son componentes diferentes.
6. Construir para cada componente un nuevo vértice y adicionarlo a una lista *NC*.
7. Retornar *NC* y *RC* (que representan los nuevos objetos a insertar y eliminar en el próximo nivel).

Resultados y discusión

Para demostrar la eficiencia de la estructura compacta multi-grafo con respecto a mantener los distintos grafos (máxima $ß_0$-semejanza, $ß_0$-semejanza) y otras estructuras (cubrimiento y la jerarquía) por separado, se muestra la comparación de los resultados temporales y costo de memoria con respecto a una implementación del ACJD, que mantiene estructuras separadas pero no almacena el grafo de $ß_0$-semejanza, el cual es calculado cada vez que es necesario. Esto implica un costo temporal adicional.

Los documentos se representaron usando el modelo de espacio vectorial tradicional (Raghavan *et al*., 1986; Aggarwal *et al*., 2012; Hernández *et al*., 2004). En este modelo los términos representan los lemas de las palabras que aparecen en los textos correspondientes. Las palabras de parada fueron eliminadas y los pesos de cada término fueron calculados mediante el TF, que es la cantidad de veces que aparece el término en el documento. Para comparar a los grupos se empleó la medida de semejanza del coseno. Y para evaluar la calidad del algoritmo de agrupamiento propuesto se utilizó la colección de documentos Reuter 2000 RCV1 (Lewis, 2004).

Todos los experimentos fueron realizados sobre una computadora personal con cuatro procesadores Genuine Intel Intel(R) Core 2 Quad CPU Q9300 @ 2.50GHz y memoria caché de 3072 KB. Con un total de 4 GB de memoria RAM. El sistema operativo usado fue el Linux Ubuntu 12.4 con una arquitectura de 64 bits.

Para realizar los experimentos se fijó un valor de $ß_0$ de 0.15 y uno de α de 0.1. Y se verificó mediante los mismos que los resultados obtenidos por nuestra variante del algoritmo ACJD y una implementación anterior existente fueran iguales.

En la Tabla 1 se muestran la cantidad de vértices del multi-grafo y cantidad de aristas de los grafos β_0-semejanza y del grafo de máxima β_0-semejanza (max-S). Como se aprecia la cantidad de aristas del grafo de β_0-semejanza es mucho mayor que las que forman parte del grafo de máxima β_0-semejanza que son las que se usan realmente para construir el agrupamiento final. Es por ello que es posible reducir la cantidad total de aristas almacenadas en el grafo de β_0-semejanza realizando una poda, teniendo en cuenta el valor de α. En la Tabla 2 se puede apreciar la disminución en el número de aristas almacenadas para el grafo de β_0-semejanza al usar un valor de $\alpha = 0.1$.

Tabla 1. Estadísticas del multi-grafo sin el parámetro α

Nivel	Cantidad de vértices	Cantidad de aristas β_0	Cantidad de aristas max-S
0	5000	4830074	4332
1	1123	300458	955
2	165	6018	142
3	20	69	15
4	4	3	2
5	1	0	0
Total	**6313**	**5136622**	**5446**

Tabla 2. Estadísticas del multi-grafo con α=0.1

Nivel	Cantidad de vértices	Cantidad de aristas β_0	Cantidad de aristas max-S
0	5000	132141	4332
1	1123	9523	955
2	165	558	142
3	20	22	15
4	4	3	2
5	1	0	0
Total	**6313**	**142247**	**5446**

Se realizó un único experimento para medir todas las características del algoritmo propuesto. Esto fue posible debido a la naturaleza dinámica del mismo, lo que permite obtener los resultados asociados a un conjunto creciente de datos en una sola corrida.

Por cada conjunto de datos procesados se obtuvo una jerarquía independiente y se generaron los valores de memoria consumida y tiempo de procesamiento. La corrida se realizó partiendo de un conjunto de 5,000 objetos e incrementándolo en pasos de 5,000 objetos.

La comparación del costo temporal es mostrada en la Figura 3. Como se aprecia, este costo es siempre menor para la implementación que se propone. Por ejemplo, aún para un bloque de 5,000 objetos el costo de usar el multi-grafo es de 195 segundos, mientras que usando grafos separados este costo es de 290 segundos. Esta diferencia se incrementa

a medida que aumenta el número de objetos insertados, llegando a tomar valores de 42 y 86 horas respectivamente para 120,000 objetos.

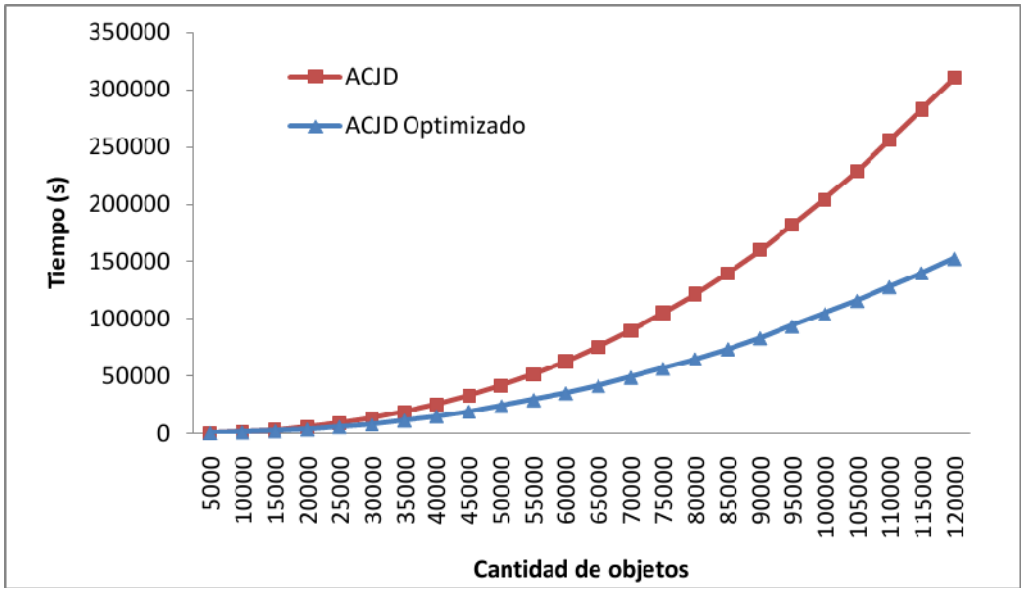

Figura 3. Comparación del costo temporal entre el algoritmo ACJD y las optimizaciones propuestas

En la Figura 4 se muestra la comparación del consumo de memoria para ambas implementaciones. Como se aprecia, con el uso del multi-grafo más el valor de α=0.1 se logra una disminución considerable del costo espacial. Esto se debe a que las distintas estructuras y grafos son solapadas en una sola estructura, y debido a que el grafo de $ß_0$-semejanza es podado eliminando a las aristas menos probables de ser usadas en al algoritmo. Por ejemplo, para un bloque de 120,000 objetos se consume con el multi-grafo aproximadamente 2 GB, mientras que con los grafos separados 3,3 GB.

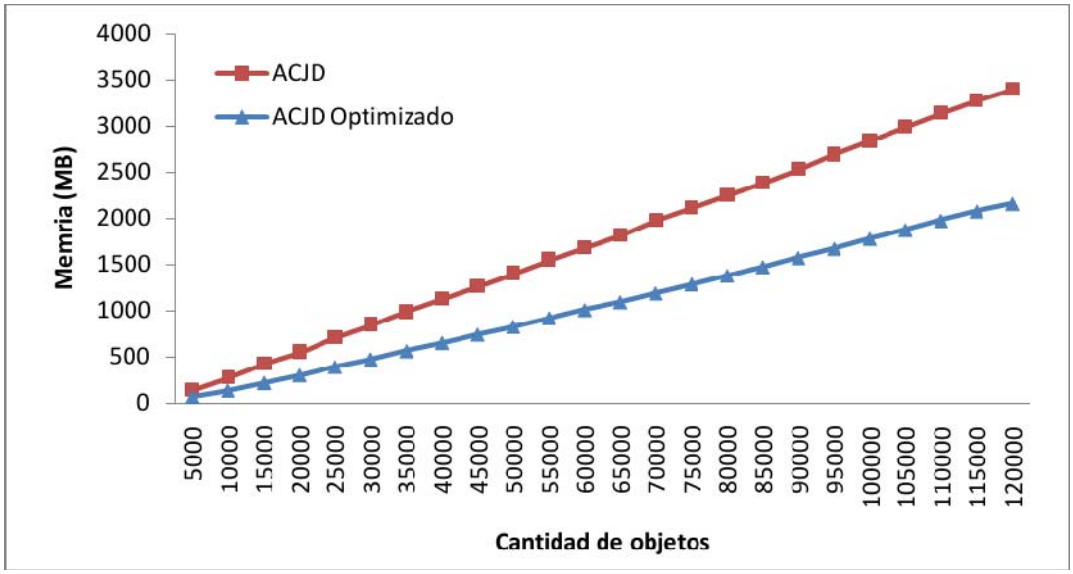

Figura 4. Comparación del consumo de memoria entre el algoritmo ACJD y las optimizaciones propuestas

Además se logró agrupar un número mayor de objetos con la nueva propuesta pues esta generó una jerarquía con 195,000 objetos, mientras que la versión anterior solamente logró agrupar 120,000. En las Figura 5 se muestran las interpolaciones de los resultados de la variante original (en la que se asuma la disponibilidad de tanta memoria adicional como sea posible), con el objetivo de realizar una comparación entre el consumo temporal real de la nueva variante (optimizaciones realizadas al ACJD) y el consumo temporal estimado de la variante original para las colecciones de mayor tamaño que la nueva variante fue capaz de procesar. Como se puede apreciar la cantidad de tiempo que la variante original necesitaría para procesar las colecciones de 195,000 objetos sería aproximadamente 250 horas mientras que con las optimaciones 114 horas.

En la Figura 6 de manera análoga al caso anterior, se estima la cantidad de memoria que la variante original necesitaría para procesar las colecciones de igual tamaño que la nueva propuesta fue capaz de procesar. En este caso el algoritmo ACJD necesitaría 5.5GB de memoria principal mientras que nuestra propuesta 3.6GB.

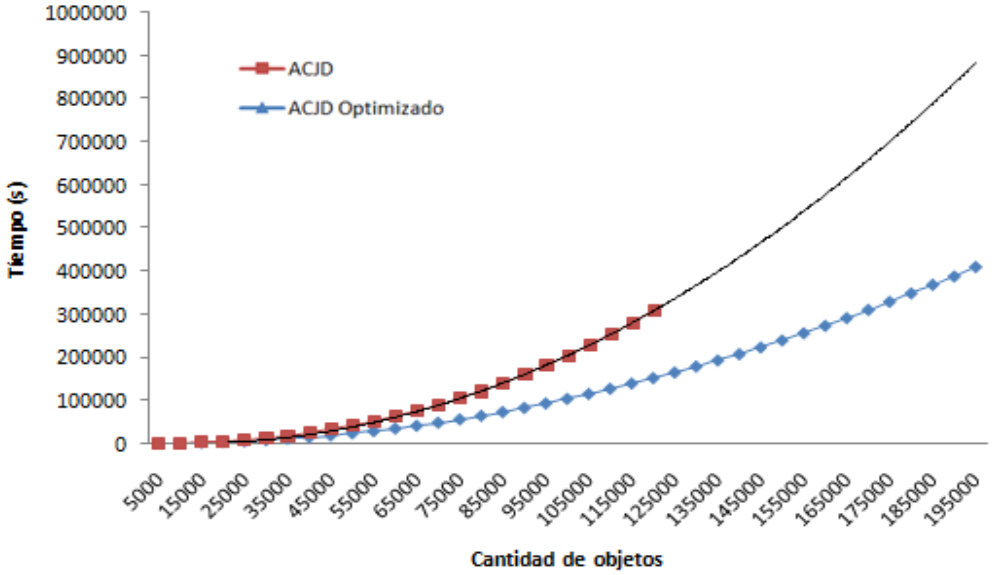

Figura 5. Comparación entre el consumo temporal real de las optimizaciones al ACJD y el consumo temporal estimado del ACJD

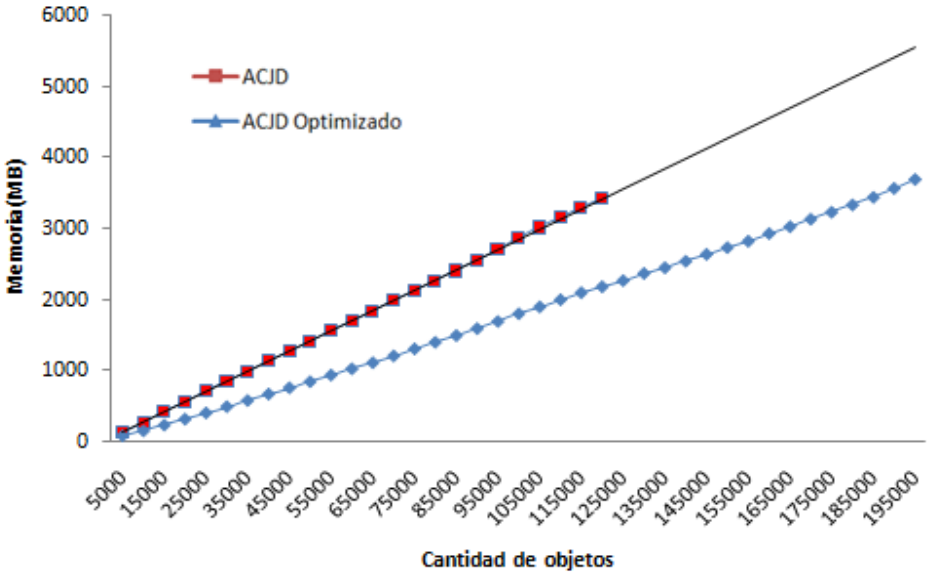

Figura 6. Comparación entre el consumo de memoria real de las optimizaciones al ACJD y el consumo de memoria estimado del ACJD

Conclusiones

En este trabajo se presentaron dos optimizaciones al algoritmo de agrupamiento ACJD, con el objetivo de agrupar un número mayor de documentos sobre un solo procesador, manteniendo la calidad de los resultados. Esto se logró siguiendo dos estrategias fundamentales. La primera consistió en el uso de una estructura de datos compacta, que permite eliminar la redundancia de los datos almacenados, aprovechando de forma óptima los recursos de memoria principal de una computadora. La segunda estrategia consistió en mantener en determinadas estructuras de datos solamente aquella información que eran mayores candidatas a ser usadas en etapas posteriores del algoritmo. Esto último se logró con el uso de una estrategia de poda que redujo grandemente el volumen de datos almacenados en estas estructuras.

Con la combinación de ambas estrategias se logró una reducción significativa en el tiempo de procesamiento y en la cantidad de memoria principal necesaria para el proceso. Se obtuvo un algoritmo que permite procesar un número mayor de documentos en un tiempo menor que con implementaciones anteriores. La utilidad práctica de este trabajo está dada en que el algoritmo forma parte de herramientas de manejo de datos más generales, y con esta nueva versión se benefician todas estas herramientas que a su vez van a poder obtener agrupamientos de más datos en menos tiempo.

Como trabajo futuro se propone el diseño de un algoritmo paralelo y distribuido, con estrategias de distribución inteligentes de los objetos, con el objetivo de que el multi-grafo sea lo más compacto posible en cada procesador, minimizando así el número de comunicaciones y disminuyendo el costo temporal total del agrupamiento. Además integrar el uso de mecanismos de virtualización de memoria principal sobre memoria secundaria, para mejorar las prestaciones en cuanto a cantidad de objetos a agrupar.

Referencias

- AGGARWAL, C., *et al.* Mining Text Data. Springer US, 2012.
- GIL, R. Algoritmos de Agrupamiento sobre Grafos y su Paralelización. Tesis doctoral en Ciencia de la Computación, Universidad Jaume I, España, 2005.
- GIL, R, *et al.* A General Framework for Agglomerative Hierarchical Clustering Algorithms. 2006. pp. 569 - 572. Vol. 2. ISBN: 0-7695-2521-0, ISSN: 1051-4651.

- GIL, R., and PONS A. Improving the Dynamic Hierarchical Compact Clustering Algorithm by Using Feature Selection, In Proceedings of the XV Iberoamerican Congress on Pattern Recognition (CIARP2010), 2010, LNCS 6419, p. 302–310.
- HERNÁNDEZ, O., *et al.* Introducción a la Minería de Datos. Madrid : PEARSON Prentice Hall, 2004.
- LEWIS D. RCV1: A New Benchmark Collection for Text Categorization Research. Journal of Machine Learning Research, 2004, 5:361-397.
- RAGHAVAN, V., *et al.* A critical analysis of Vector Space Model for Information Retrieval, Journal of the American Society on Information Science, 1986, Vol. 37, pp 279-287.
- RUIZ, S., *et al.* Enfoque lógico combinatorio al reconocimiento de patrones. México, 1999. ISBN: 970-18-2059-2.