Algoritmo incremental de agrupamiento con traslape para el procesamiento de grandes colecciones de datos

Lázaro Janier González-Soler
jsoler@cenatav.co.cu
Centro de Aplicaciones de Tecnologías de Avanzada
Airel Pérez-Suárez
asuarez@cenatav.co.cu
Centro de Aplicaciones de Tecnologías de Avanzada
Leonardo Chang-Fernández
lchang@cenatav.co.cu
Centro de Aplicaciones de Tecnologías de Avanzada

RESUMEN

Existen diversos problemas en el Reconocimiento de Patrones y en la Minería de Datos que, por su naturaleza, consideran que los objetos pueden pertenecer a más de una clase o grupo. DClustR es un algoritmo dinámico de agrupamiento con traslape que ha mostrado, en tareas de agrupamiento de documentos, el mejor balance entre calidad de los grupos y eficiencia entre los algoritmos dinámicos de agrupamiento con traslape reportados en la literatura. A pesar de obtener buenos resultados, DClustR puede ser poco útil en aplicaciones que trabajen con grandes colecciones de documentos, debido a que tiene una complejidad computacional $O(n^2)$ y a la cantidad de memoria que utiliza para el procesamiento de las colecciones. En este trabajo se presenta una versión paralela basada en GPU del algoritmo DClustR, llamada CUDA-DClus, para mejorar la eficiencia de DClustR en aplicaciones que lidien con largas colecciones de documentos. Los experimentos fueron realizados sobre varias colecciones estándares de documentos y en ellos se muestra el buen rendimiento de CUDA-DClus en términos de eficiencia y consumo de memoria.

PALABRAS CLAVE : Agrupamiento, Agrupamiento con traslape, Computación en GPU, Minería de Datos.

ABSTRACT

There are several problems in Pattern Recognition and Data Mining that, by its inherent nature, consider that objects could belong to more than one class o cluster. DClustR is a dynamic overlapping clustering algorithm that has shown, in the task of document clustering, the better tradeoff between quality of the clusters and efficiency among the exiting dynamic overlapping clustering algorithms. Despite the good achievements attained by DClustR, this could be less useful in applications dealing with a large number of documents, due to it has a computational complexity of $O(n^2)$ and the amount of memory that it uses in order to the processing of collections. In this paper, a GPU-based parallel algorithm of DClustR, named CUDA-DClus is proposed in order to enhance the efficiency of DClustR, in problems dealing with a large number of documents. The experimental

evaluation conducted over several standard document collections showed the CUDA-DClus better performance in terms of efficiency and memory consumption.

INTRODUCCIÓN

El agrupamiento es una técnica del Aprendizaje de Máquina y la Minería de Datos que ha sido utilizada de diversos contextos (Bae et al., 2010). Esta técnica permite estructurar una colección de objetos en grupos o clases, donde los objetos pertenecientes a un mismo grupo son más similares respecto a los objetos pertenecientes a diferentes clases (Jain et al., 1999).

Un problema que recientemente ha recibido mucha atención, es el desarrollo de los algoritmos de agrupamiento con traslape. Los grupos construidos por estos algoritmos permiten que un objeto pueda pertenecer a más de un grupo. Uno de los algoritmos que ha mostrado el mejor balance entre la calidad de los grupos y eficiencia, entre los algoritmos dinámicos de agrupamiento con traslape reportados en la literatura, es DClustR (Pérez-Suárez et al., 2013) (del inglés Dynamic Overlapping Clustering based on Relevance). Este algoritmo introduce una estrategia para actualizar eficientemente el agrupamiento después de múltiples adiciones y/o eliminaciones en la colección, haciéndolo útil en aplicaciones donde las colecciones cambian frecuentemente. A pesar de que DClustR ha obtenido buenos resultados en tareas de agrupamiento de documentos, este tiene una complejidad algorítmica de $O(n^2)$. Esto implica que cuando las colecciones crecen demasiado, el tiempo que DClustR utiliza para procesar todos los cambios aumenta, haciéndolo poco útil en aplicaciones reales. Además, cuando las colecciones crecen, la memoria utilizada por DClustR para almacenar los datos también aumenta, por lo que puede resultar poco útil en aplicaciones que lidien con grandes colecciones de documentos.

Una técnica que ha sido muy utilizada en los últimos años para acelerar tareas de cómputos es la computación paralela y específicamente, la computación en GPU. La GPU es un dispositivo que fue inicialmente diseñado para el procesamiento de algoritmos pertenecientes al mundo gráfico, pero debido a su bajo costo, alto nivel de paralelismo y su optimización en las operaciones con punto flotante, este ha sido usado en muchas aplicaciones reales que lidian con un largo número de objetos.

La principal contribución de este trabajo es el diseño e implementación de una variante paralela basada en GPU del algoritmo DClustR, llamado CUDA-DClus, el cual mejora la eficiencia de DClustR en problemas que lidien con largas colecciones de documentos como por ejemplo análisis de noticias, organización de información, identificación de perfiles, entre otras. Se introdujo también una estrategia para construir y actualizar incrementalmente las componentes conexas presentadas en un grafo, permitiendo que CUDA-DClus minimice la cantidad de memoria utilizada para procesar toda la colección. Es importante señalar que en el desarrollo de CUDA-DClus se analizó únicamente la adición de nuevos objetos a la colección debido a que este es el caso que hace que la colección crezca y la cual puede dificultar que DClustR se utilice en aplicaciones reales que lidien con grandes colecciones.

El resto de este capítulo está organizado de la siguiente manera: en la sección 2, se describe brevemente el algoritmo DClustR. En la sección 3, se describe la versión paralela CUDA-DClus. Los resultados experimentales que muestran el rendimiento del algoritmo paralelo

sobre varias colecciones de documentos, es presentado en la sección 4. Finalmente, las conclusiones son presentadas en la sección 5.

DCLUSTR ALGORITHM

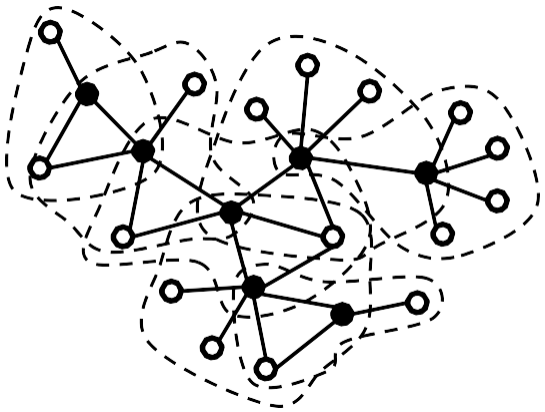
En esta sección se presenta unos conceptos preliminares, así como una breve descripción del funcionamiento del algoritmo DClustR (Pérez-Suárez et al. 2013) para procesar las adiciones de nuevos objetos en la colección. Todas las definiciones y ejemplos presentadas en esta sección fueron tomadas (Pérez-Suárez et al. 2013).

Sea $O = \{o_1, o_2, ..., o_n\}$ una colección de objetos, $\beta \in [0,1]$ un umbral de semejanza y $S: O \times O \rightarrow \mathbb{R}$ una función de semejanza simétrica. Sea $\widetilde{G_\beta} = \langle V, \widetilde{E_\beta}, S \rangle$ un grafo de β-semejanza pesado en el cual $V = O$ y $\exists (u,v) \in \widetilde{E_\beta}, v\ G\ u \Leftrightarrow S(u,v) \geq \beta$.

Sea $\widetilde{G_\beta} = \langle V, \widetilde{E_\beta}, S \rangle$ un grafo de β-semejanza pesado. Un sub-grafo pesado en forma de estrella en $\widetilde{G_\beta}$, denotado por $G^\star = \langle V^\star, E^\star, S \rangle$, es un sub-grafo que tiene un vértice $c \in V^\star$ de manera tal que existe una arista entre c y los restantes vértices en $V^\star$. El vértice c es llamado centro del sub-grafo y los restantes vértices son llamados satélites.

Sea $\widetilde{G_\beta} = \langle V, \widetilde{E_\beta}, S \rangle$ un grafo de β-semejanza pesado y $W = \{G_1^\star, G_2^\star, ..., G_k^\star\}$ un conjunto de grafos, de manera tal que cada $G_i^\star$ es un sub-grafo de $\widetilde{G_\beta}$. El conjunto W es un cubrimiento de $\widetilde{G_\beta}$ si y solo si $V = \bigcup_{i=1}^{k} V_i^\star$. Además, un vértice v está cubierto por un sub-grafo $G_i^\star \in W$ si $v \in V_i^\star$ (ver Figura 1).

Figura 1: Conjunto de sub-grafos en forma de estrella que conforman los grupos construidos por DClustR. Los vértices marcados en negro son los centros de cada sub-grafo. Las líneas discontinuas encierran cada grupo.



DClustR utiliza una estrategia eficiente para construir y mantener actualizado el conjunto W cuando ocurren adiciones o eliminaciones de objetos. Cada $G_i^\star \in W$ es tratado como un grupo por DClustR. La estrategia usada por DClustR para actualizar los grupos previamente construidos cuando ocurren múltiples cambios consiste en detectar las componentes conexas de $\widetilde{G_\beta}$ que fueron afectadas por los cambios, para posteriormente actualizarlas.

Para procesar las adiciones de objetos en $\widetilde{G_\beta}$, DClustR, en un primer paso, calcula la semejanza entre los nuevos vértices y los existentes en $\widetilde{G_\beta}$, añadiendo una arista entre dos vértices si su semejanza es mayor o igual que β. Es importante mencionar que, aunque DClustR es un algoritmo dinámico, en nuestro trabajo se analizó únicamente la adición de objetos a la colección debido a que esta operación incrementa la cantidad de objetos en la colección y hace que DClustR sea poco útil en aplicaciones reales que lidien con largas colecciones de objetos.

Sea $G' = \langle V', E' \rangle$ una componente conexa afectada por la adición de un objeto. Para actualizar el cubrimiento de G', DClustR realiza 4 pasos fundamentales. En el primer paso, se recorre cada vértice de G' y se construye el conjunto de vértices $C' = \{c_1, c_2, \ldots, c_k\}$ que forman el cubrimiento previo de G' y el conjunto V_s', que contiene los vértices con relevancia mayor que cero que pertenecen a G' y no a C'. Los vértices $c_i \in C'$ son los centros de los sub-grafos en forma de estrella que componen G'. La relevancia de un vértice se calcula como el promedio entre su densidad relativa y su compacidad relativa. La densidad relativa de un vértice v es la cantidad de vértices adyacentes a v que tienen grado menor o igual que v, dividido entre el grado de v. La compacidad relativa se calcula como la razón entre el número de vértices $u \in \text{adyacentes}[v]$ que cumplen que $\text{AIS}(v) \geq \text{AIS}(u)$ y la cantidad de adyacentes a v; $\text{AIS}(v)$ es el promedio de semejanza entre v y sus adyacentes. Una vez construidos los conjuntos C' y V_s', en el segundo paso se construye una lista de candidatos L' con aquellos vértices que pueden ser añadidos a C'; durante esta operación se pueden eliminar algunos vértices de C'. En el tercer paso, se actualiza C' insertando algunos vértices de L'. Finalmente, en el cuarto paso se filtra el conjunto C' eliminando algunos vértices de C' identificados como no útiles. Finalmente, el conjunto actualizado determina el agrupamiento de G'.

ALGORITMO PARALELO CUDA-DCLUS

Como se mencionó en la sección 1, a pesar que DClustR alcanza buenos resultados en tareas de agrupamiento de documentos, este tiene un complejidad algorítmica de $O(n^2)$ por lo tanto, puede ser menos útil en aplicaciones que manejen largos conjuntos de documentos. Motivado por este hecho, en esta sección se propone una implementación masivamente paralela en CUDA del algoritmo DClustR para mejorar la eficiencia de DClustR en los problemas mencionado anteriormente. La versión paralela llamada CUDA-DClus aprovecha los beneficios de la GPU como, por ejemplo, el ancho de banda en la comunicación entre la CPU y la GPU y la jerarquía de memoria de la GPU.

Aunque DClustR fue propuesto como un algoritmo de agrupamiento de propósito general, la versión paralela propuesta en este trabajo está específicamente diseñada para el procesamiento de documentos. Este contexto de aplicación es el mismo en el cual DClustR fue evaluado y también un contexto donde es muy común el procesamiento de largas colecciones de documentos. En el contexto del procesamiento de documentos, CUDA-DClus utiliza la medida del coseno (Berry and Castellanos 2004) para calcular la semejanza entre dos documentos. La medida del coseno ha sido la función más utilizada para este propósito (Gil-García and Pons-Porrata 2010) y define la semejanza entre dos documentos d_i y d_j como:

$$S(d_i, d_j) = \cos(d_i, d_j) = \frac{\sum_{k=1}^{m} d_i(k) * d_j(k)}{||d_i|| \cdot ||d_j||},$$

donde $d_i(k)$ y $d_j(k)$ son el peso del término k en la descripción de los documentos d_i y d_j respectivamente; $||d_i||$ y $||d_j||$ son las normas de los documentos d_i y d_j respectivamente.

En los experimentos realizados sobre varias colecciones de documentos, se verificó que la actualización del grafo después de los cambios, así como el cálculo de la relevancia son las etapas que consumen más tiempo de procesamiento de DClustR. La primera etapa consume el 99% del tiempo de procesamiento del algoritmo y la segunda etapa es parte del 1% restante del tiempo de procesamiento del algoritmo. La primera etapa fue implementada en CUDA por el algoritmo CUDA-DClus. En este caso, es importante señalar que la detección de las componentes conexas afectadas por los cambios es una tarea altamente consumidora de memoria del sistema, por lo tanto, se propuso una estrategia para resolver este problema. Finalmente, es importante mencionar que debido a que en este trabajo se manejan largas colecciones de documentos, CUDA-DClus implementó únicamente la operación de adición de nuevos documentos. Esta operación es la única que incrementa el tamaño de las colecciones.

Sea $D = \{d_1, d_2, \ldots, d_n\}$ una colección de documentos donde cada $d_i \in D$ se describe por un conjunto de términos. Sea $T = \{t_1, t_2, \ldots, t_m\}$ el conjunto de términos que describen al menos un documento en D. La implementación de CUDA-DClus asume que cada documento $d_i \in D$ está representado por dos vectores T_{d_i} y W_{d_i}. El primero contiene la posición en T de los términos que describen a d_i y el otro, la frecuencia que tiene cada uno de estos términos en el documento d_i.

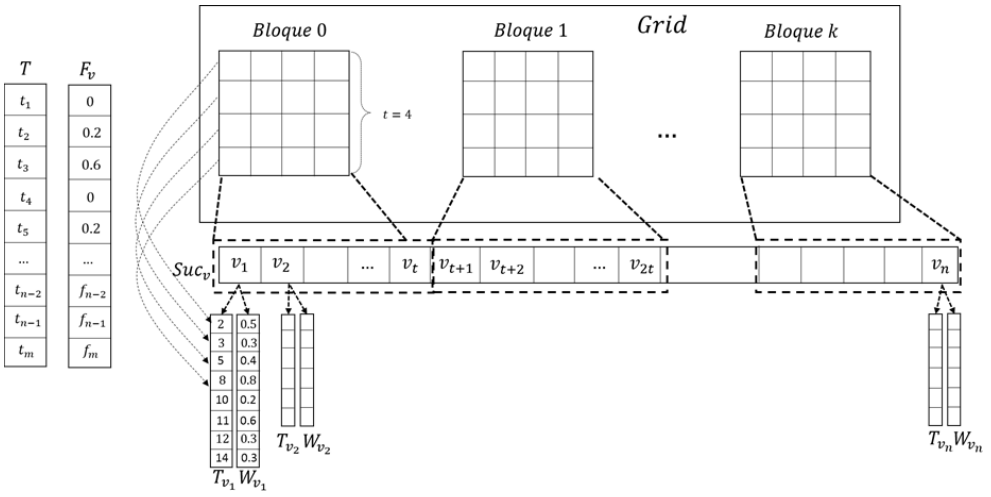
Analizando la definición de la medida del coseno, se puede observar que: (1) el numerador es una suma de productos independientes los cuales pueden ser calculados simultáneamente, (2) la norma de un documento puede ser calculada mientras que el documento es leído, lo que significa que este proceso no requiere un tiempo extra. Basado en este hecho, CUDA-DClus para acelerar la construcción del grafo paralelizó el cálculo de la semejanza entre cada $d_i \in D$ y el conjunto $Suc_{d_i} = \{d_j \in D: j > i\}$ y además, la semejanza entre cada par de documentos.

Para llevar a cabo la idea mencionada anteriormente, CUDA-DClus construye un Grid compuesto por k bloques cuadrados, siendo $k = \frac{n}{t} + 1$. Cada bloque define una matriz de memoria compartida de tamaño t; siendo t el máximo valor permitido por la arquitectura del GPU para representar una matriz de memoria compartida por bloque. Un grid es una configuración lógica de hilos en la GPU. El uso de la matriz de memoria compartida permite que CUDA-DClus no acceda constantemente a la memoria global del GPU para calcular la semejanza entre dos vértices; la memoria compartida por bloque es mucho más rápida que la memoria global.

Cuando CUDA-DClus calcula simultáneamente la semejanza entre d_i (a partir de ahora v) y el conjunto Suc_{d_i} (Suc_v), CUDA-DClus primero construye un vector F_v con tamaño m. Este vector tiene todas las posiciones con valor 0 menos aquellos elementos asociados a las posiciones almacenadas en T_v, las cuales contienen sus respectivos pesos almacenados en W_v. Un vez construido F_v, el conjunto Suc_v es particionado en k subconjuntos y cada uno de estos subconjuntos es asignado a un bloque del Grid. En este contexto, una columna de un bloque representa un vértice $u \in Suc_v$ y el conjunto de filas que representan la columna

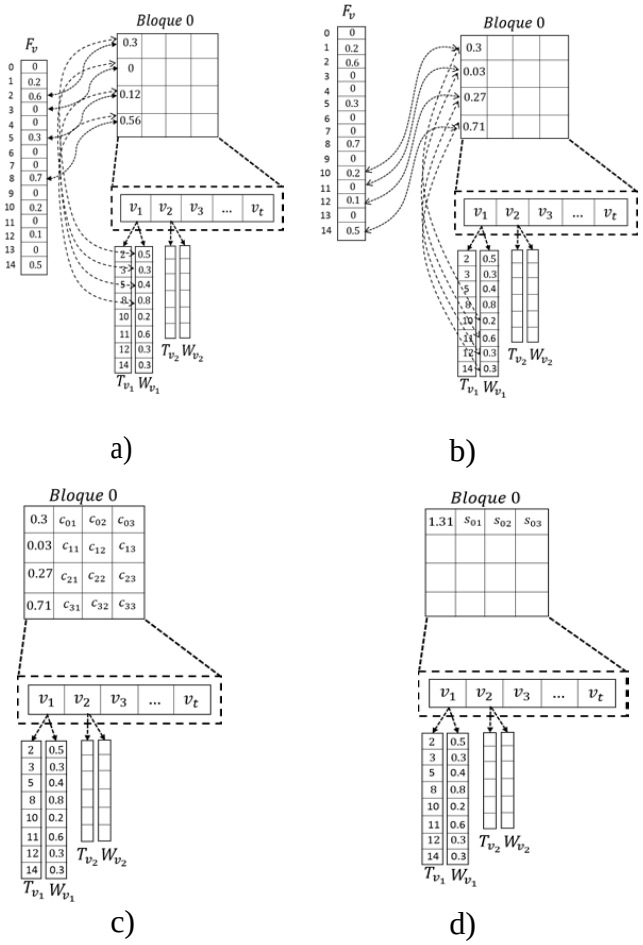
asociada a u señalan los términos que describen a u; de esta manera la fila j — ésima señala el j — ésimo término que describe al vértice u. La Figura 2 muestra lo explicado anteriormente. Este ejemplo muestra cómo se distribuye cada vértice en los bloques que conforman el Grid.

Figura 2: Ilustración de cómo CUDA-DClus divide el conjunto Suc_v ***en subconjunto para posteriormente asignarlos en cada bloque.***



Cada hilo asociado a una columna i — ésima en un bloque se encarga de calcular la semejanza entre v y el vértice asociado a la columna; de aquí en lo adelante referido como v_i. Para calcular la semejanza entre un vértice v y v_i el hilo de cada celda en la columna calculará el producto entre la frecuencia asociada al término representado por dicho hilo y la frecuencia que tiene el mismo término en el vector F_v (Figura 3a)). El resultado obtenido por cada hilo se almacenará en la matriz de memoria compartida correspondiente al bloque. Si un vértice asignado a un bloque lo describe más de t términos, entonces se aplica una técnica llamada *Tiling* (Sanders and Kandrot 2010) (Figura 3b)). Esta técnica consiste en dividir un conjunto de datos en pequeños subconjuntos de manera tal que cada subconjunto pueda ser almacenado en la matriz de memoria compartida. Apoyados en esta idea, CUDA-DClus mueve iterativamente los hilos de una columna asociados a un subconjunto de términos, hacia el siguiente subconjunto de términos no procesados, reutilizando la información previamente almacenada en la matriz de memoria compartida. Una vez calculado todos los productos en un bloque (Figura 3c)) se aplica una técnica llamada *Reduction* sobre cada columna de la matriz de memoria compartida asociada al bloque (Figura 3d)). *Reduction* (Sanders and Kandrot 2010) es una técnica muy útil para calcular simultáneamente la suma acumulativa en un vector con un tamaño potencia del valor 2. Una vez aplicado la técnica *Reduction* para cada columna en todos los bloques, la primera fila de cada bloque tendrá el valor de semejanza entre el vértice v y los vértices del conjunto Suc_v. Una vez concluidos todos los cálculos, los resultados son normalizados y copiados hacia la memoria del sistema. Durante este proceso se puede calcular también el $AIS(d_i)$, sin afectar el costo computacional de la etapa.

Figura 3: Diferentes etapas del cálculo de la semejanza entre el vértice v ***y un subconjunto de*** Suc_v ***asignado al primer bloque del*** Grid***, a) producto de los primeros*** t ***términos de*** v ***respecto al vértice asociado con la primera columna del bloque 0, b) Redistribución de hilos aplicando la técnica Tiling para continuar el producto de términos, c) bloques con todas los productos calculados y d) semejanza del vértice*** v ***con cada vértice*** v_i ***almacenada en la primera fila del bloque 0 después de aplicar la técnica Reduction.***



Siguiendo la idea anterior, CUDA-DClus actualiza el grafo con la adición de nuevos documentos a la colección. Sea $I = \{i_1, i_2, \ldots, i_t\}$ el conjunto de documentos que serán añadidos a $O = \{o_1, o_2, \ldots, o_n\}$ y sea $\widetilde{G_\beta} = \langle V, \widetilde{E_\beta}, S \rangle$ el grafo asociado a la colección O. Para actualizar $\widetilde{G_\beta}$ con los nuevos documentos de I, CUDA-DClus ejecuta tres fases fundamentales. En la primera fase, CUDA-DClus construye para cada documento en I un vértice en el grafo. Posteriormente, en la segunda fase se recorre I y para cada documento $i_p \in I$, se calcula en paralelo la semejanza entre i_p y los documentos del conjunto O usando la estrategia paralela propuesta anteriormente.

Como se mencionó en la sección anterior, DClustR primero, detecta las componentes conexas afectadas por la adición de nuevos documentos para actualizar el agrupamiento de esas componentes y por consecuente, el agrupamiento de toda la colección. Las

componentes conexas afectadas son aquellas que contienen al menos un vértice añadido. Cada vez que se añade un nuevo vértice a $\widetilde{G_\beta}$ se calcula la semejanza entre dicho vértice y los vértices de $\widetilde{G_\beta}$ para crear las respectivas aristas. Una vez finalizado lo anterior, DClustR necesita construir desde cero cada componente conexa afectada para actualizar su cubrimiento. Para reducir la cantidad de información que DClustR almacena en memoria, CUDA-DClus propone una estrategia para representar el grafo $\widetilde{G_\beta}$ usando una lista de componentes conexas parciales llamada L_{PCC} y otras dos listas. La primera lista llamada V, contiene los vértices del grafo en el orden que fueron añadidos a $\widetilde{G_\beta}$. La otra lista llamada PC_V, contiene el índice de las componentes conexas parciales a las que pertenece cada vértice. Esta nueva representación permite que CUDA-DClus no necesite reconstruir las componentes conexas afectadas cada vez que la colección cambie y, además, permite, en un orden de complejidad constante, mantener actualizadas las componentes afectadas cada vez que se añade un nuevo vértice al grafo. Además, esta representación permite que CUDA-DClus solamente utilice la memoria del sistema para almacenar únicamente las componentes conexas afectadas por la adición de nuevos vértices.

Sea $\widetilde{G_\beta} = \langle V, \widetilde{E_\beta}, S \rangle$ un grafo de β-semejanza pesado que representa a la colección de documentos. Una componente conexa parcial (PCC) en $\widetilde{G_\beta}$ es un sub-grafo conexo inducido por un subconjunto de vértices de $\widetilde{G_\beta}$. Una PCC está representada por dos listas. Una lista contiene el índice en $\widetilde{G_\beta}$ de los vértices que pertenecen a esa componente y la otra contiene la lista de adyacencia de los vértices anteriores.

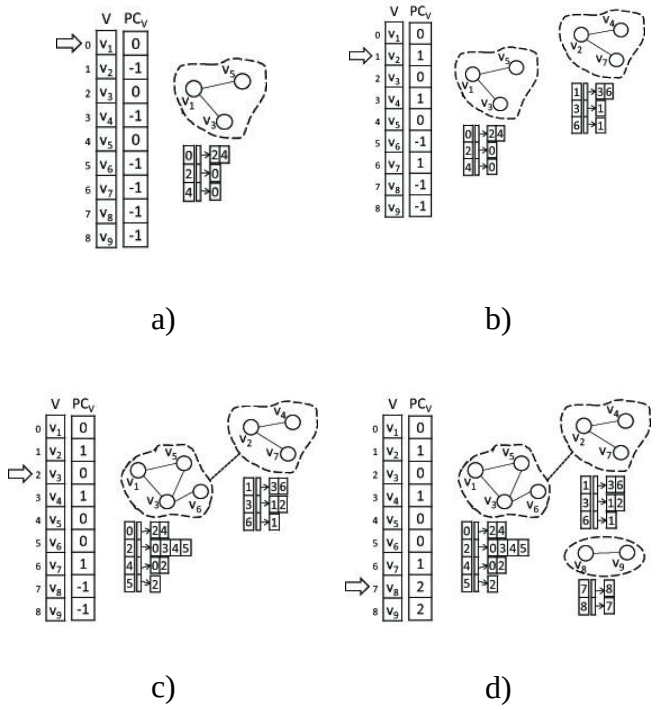
La estrategia que CUDA-DClus utiliza para construir la lista de componentes conexas parciales que representan a $\widetilde{G_\beta}$ sigue los siguientes pasos. En la primera etapa, CUDA-DClus añade para cada documento en la colección un vértice en el grafo; PC_V tiene todos sus elementos con valor $-\infty$, lo que significa que los vértices no pertenecen a ninguna PCC todavía. En el segundo paso, CUDA-DClus procesa cada vértice $v_i \in V$ y crea una PCC con v_i si este no pertenece a ninguna PCC. Cuando un vértice es añadido a una PCC, el valor del índice que esta PCC tiene en la lista L_{PCC} es almacenado en la entrada i-ésima de la lista PC_V; la entrada i-ésima está asociada con el vértice v_i. En un tercer paso, CUDA-DClus calcula simultáneamente la semejanza entre el vértice v_i y el conjunto Suc_{v_i} utilizando la estrategia paralela propuesta anteriormente. Una vez que la semejanza ha sido calculada, CUDA-DClus visita cada vértice $u \in Suc_{v_i}$ y si $S(v_i, u) \geq \beta$ y u no pertenece a ninguna PCC, entonces u es agregado a la PCC de v_i y la lista de adyacencia que representa a la PCC es actualizada para indicar que ambos vértices son adyacentes. En otro caso, si u pertenecía a un PCC diferente a la PCC de v_i entonces, únicamente son actualizadas las listas de adyacencia de ambos vértices. Cuando dos vértices adyacentes pertenecen a diferentes componentes conexas parciales, se dice que estas componentes conexas parciales están enlazadas. Es importante señalar, que en todos los casos se actualiza el valor de los índices en la lista PC_V para indicar a que PCC pertenece cada vértice. La Figura 4 muestra los pasos para representar el grafo $\widetilde{G_\beta}$ utilizando listas de componentes conexas parciales.

RESULTADOS EXPERIMENTALES

En esta sección, se presentan los resultados de varios experimentos donde se evalúa el rendimiento del algoritmo CUDA-DClus. Los experimentos se realizaron sobre 6

colecciones de documentos y se enfocaron en: i) evaluar la aceleración alcanzada por CUDA-DClus respecto al algoritmo DClustR, y ii) evaluar el costo espacial de la versión paralela respecto a la secuencial. Todos los algoritmos están implementados en C++; el código del DClustR fue obtenido de sus autores. La implementación del algoritmo CUDA-DClus se realizó utilizando el CUDA toolkit 5.5. Los experimentos se realizaron sobre una PC Core i7-4770 @ 3.40GHz, 8GB RAM, PCI express NVIDIA GeForce GT 635, con 1GB DRAM.

Figura 4: Ejemplo para construir $\widetilde{G_Q}$ ***utilizando componentes conexas parciales, a) PCC con los vértices semejantes a*** v_1***, b) construcción de la PCC del vértice*** v_2***, c) PCCs enlazadas por el vértice*** v_3 ***y*** v_4***, d) construcción de la PCC del vértice*** v_8***.***



Las colecciones de documentos utilizadas en nuestros experimentos fueron construidas a partir de dos corpus de referencia, comúnmente usadas en el agrupamiento de documentos: Reuters-v2 y TDT2. La Reuters-v2 puede ser obtenida de http://kdd.ics.uci.edu y la TDT2 puede ser obtenida de http://www.nist.gov/speech/tests/tdt.html. A partir de los dos corpus de referencia, se construyeron seis colecciones. Las características de las colecciones son mostradas en la Tabla 1.

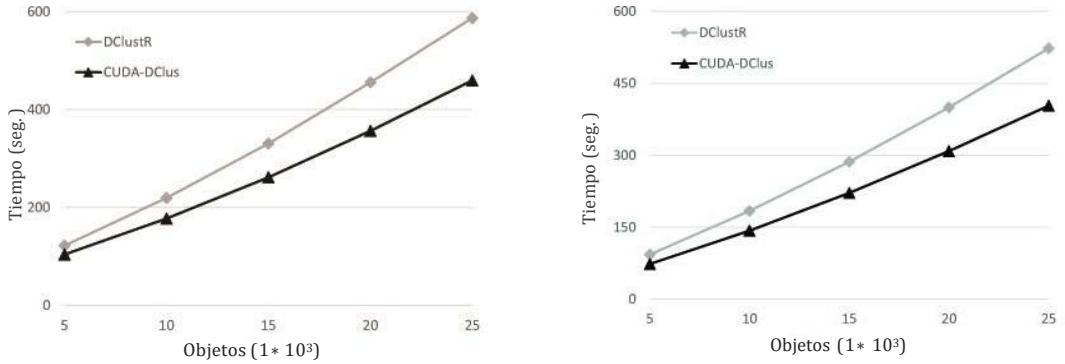
Tabla 1 Características de las colecciones utilizadas en nuestros experimentos

Colección	# Documentos	# Términos	Términos/Documentos
tdt-v5K	5 000	33 312	22
tdt-v10K	10 000	46 301	31
tdt-v15K	15 000	55 258	39
tdt-v20K	20 000	62 725	45
tdt-v25K	25 000	69 523	50
Reu-v50K	50 000	74 720	64

En nuestros experimentos, los documentos fueron representados utilizando el Modelo de Espacio Vectorial (Salton et al. 1975). Los términos indexados de los documentos representan los lexemas de las palabras que ocurren al menos una vez en la colección; esos lexemas fueron extraídos de los documentos utilizando *Tree-tagger*[1]. Las palabras vacías tales como artículos, preposiciones y adverbios fueron eliminadas. Los términos indexados de cada documento fueron estadísticamente pesados usando el valor de su frecuencia en el documento. Finalmente, la medida del coseno fue utilizada para calcular la semejanza entre dos documentos (Berry and Castellanos 2004).

Como se mencionó anteriormente, el primer experimento evaluó la aceleración del algoritmo CUDA-DClus respecto al algoritmo DClustR. Para realizar esta prueba se realizó inicialmente el agrupamiento de la colección Reu-v50K utilizando ambos algoritmos y entonces, se midió el tiempo que cada algoritmo emplea para actualizar el actual agrupamiento cada vez que N documentos son incrementalmente añadidos a la colección. En los experimentos se utilizó como umbrales $\beta = 0.25$ y $\beta = 0.35$ y se utilizó las colecciones tdt-v5K, tdt-v10K, tdt-v15K, tdt-v20K y tdt-v25K para incrementalmente actualizar el agrupamiento inicial, siendo N=5000; este valor es más grande que los usados para evaluar el algoritmo DClustR (Pérez-Suárez et al. 2013). Teniendo en cuenta que el agrupamiento construido por ambos algoritmos depende del orden en que se procesen los datos, se ejecutó cada algoritmo diez veces sobre cada colección y cada configuración de parámetros, variando el orden de los documentos en cada colección. La Figura 5 muestra el tiempo promedio de ejecución que cada algoritmo utiliza para actualizar el agrupamiento inicial, para cada configuración de parámetros.

Figura 5: Tiempo que emplean DClustR y CUDA-DClus para actualizar el agrupamiento inicial, usando $Q = 0.25$ ***y*** $Q = 0.35$**.**



(a) $\beta = 0.25$ (b) $\beta = 0.35$

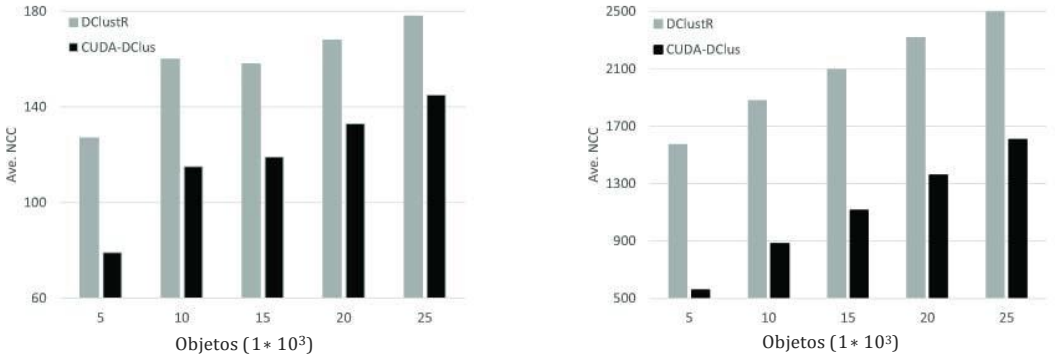
Como se puede observar en la Figura 5, CUDA-DClus utiliza, para cada configuración de parámetros, menos tiempo que DClustR para actualizar el agrupamiento, cuando ocurren múltiples adiciones sobre la colección inicial. Además, se puede observar como a medida que crece el tamaño de las colecciones, CUDA-DClus se comporta más rápido que DClustR, por lo tanto, se puede decir que CUDA-DClus escala su velocidad a medida que el tamaño de las colecciones crece.

Aunque la complejidad espacial de ambos algoritmos es $O(|V| + |\widetilde{E_\beta}|)$, la estrategia utilizada por CUDA-DClus permite reducir la cantidad de memoria necesitada para actualizar el agrupamiento cada vez que la colección cambia. En el segundo experimento se

[1] http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger

evaluó la cantidad de memoria usada por CUDA-DClus contra la cantidad que usada por DClustR cuando se procesan las actualizaciones mostrada en el primer experimento. Cabe destacar que la cantidad de componentes conexas cargadas en memoria es directamente proporcional a la cantidad de memoria utilizada por cada algoritmo. Basado en esto, la Figura 6 muestra el promedio de componentes conexas que cada algoritmo carga en memoria del sistema cuando procesa los cambios presentados en la Figura 5, para cada configuración de parámetros.

Figura 6: Promedio de la cantidad de componentes conexas cargadas por CUDA-DClus y DClustR en memoria del sistema cuando actualizan el agrupamiento inicial, usando $Q = 0.25$ ***y*** $Q = 0.35$**.**



(a) $\beta = 0.25$ (b) $\beta = 0.35$

Los resultados presentados en la Figura 6 muestran que la estrategia propuesta por CUDA-DClus para representar el grafo hace que CUDA-DClus utilice menos memoria del sistema que el algoritmo DClustR. Esta característica, más el hecho que CUDA-DClus es más rápido DClustR, permite que CUDA-DClus sea más adecuado para aplicaciones que procesen largas colecciones de documentos.

CONCLUSIONES

En este trabajo se presentó una versión paralela basada en GPU del algoritmo DClustR, llamada CUDA-DClus, específicamente adaptada para el agrupamiento de documentos. CUDA-DClus propuso una estrategia para acelerar el paso que consume más tiempo de procesamiento en DClustR. Además, se propuso una nueva estrategia para representar el grafo $\widetilde{G_\beta}$ que DClustR utiliza para representar la colección de documentos. Esta nueva representación permite que CUDA-DClus utilice menos cantidad de memoria del sistema y también evita reconstruir las componentes conexas afectadas cada vez que la colección cambie. Además, mantiene las componentes conexas actualizadas después de los cambios de la colección sin utilizar un costo extra de tiempo.

Los experimentos se realizaron sobre colecciones de documentos y se enfocaron en evaluar la aceleración y el costo espacial de la versión paralela respecto al algoritmo original. A partir de los resultados experimentales se puede concluir, que el algoritmo CUDA-DClus mejora la eficiencia del algoritmo DClustR, en problemas que manejan largas colecciones de documentos. Los experimentos mostraron que CUDA-DClus puede procesar la misma cantidad de documentos que DClustR utilizando menos memoria del sistema. Esta observación se debe a la cantidad de componentes conexas que utiliza CUDA-DClus

respecto a la cantidad utilizada por DClustR. Basado en lo anterior, se puede concluir que CUDA-DClus mejora la eficiencia de DClustR en problemas que lidian con un largo número de documentos. Este algoritmo puede ser muy útil en aplicaciones como análisis de noticias, segmentación de tópicos, entre otras. Es importante mencionar que, aunque la propuesta paralela presentada en este trabajo fue desarrollada para el procesamiento de documentos con el propósito de utilizar la medida del coseno, la estrategia propuesta paralela para construir y actualizar el grafo puede ser extendida fácilmente para trabajar con otras funciones de semejanzas como la distancia euclidiana y la de manhattan.

REFERENCIAS

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

Berry, M. W. y Castellanos, M. (2004). Survey of text mining. Computing Reviews, 45(9): 548.

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

Jain, A. K., Murty, M. N. y Flynn, P. J. (1999). Data clustering: a review. ACM Computing surveys (csur), 31(3): 264-323.

Pérez-suárez, A., Martínez-trinidad, J. F., Carrasco-Ochoa, J. A. y Medina-Pagola, J. E. (2013). An algorithm based on density and compactness for dynamic overlapping clustering. Pattern Recognition, 46(11): 3040-3055.

Salton, G., Wong, A. y Yang, Ch. (1975). A vector space model for automatic indexing. Communications of the ACM, 18(11): 613-620.

Sanders, J. y Kandrot, E. (2010). Cuda by example: an introduction to general-purpose gpu programming, Addison-Wesley Professional.