EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Mejora del algoritmo Fuzzy C-Means mediante el paradigma de programación distribuida y/o paralela

presentada por

Ing. César David Rey Figueroa

como requisito para la obtención del grado de

Maestría en Ciencias de la Computación

Director de tesis

Dr. Joaquín Pérez Ortega

Codirectora de tesis

Dra. María Yasmin Hernández Pérez

Cuernavaca, Morelos, México. Agosto de 2023.

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

Cuernavaca, Mor., 09/agosto/2023
No. De Oficio: SAC/130/2023
Asunto: Autorización de impresión de tesis

CÉSAR DAVID REY FIGUEROA
CANDIDATO AL GRADO DE MAESTRO EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"MEJORA DEL ALGORITMO FUZZY C-MEANS MEDIANTE EL PARADIGMA DE PROGRAMACIÓN DISTRIBUIDA Y/O PARALELA"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Conocimiento y tecnología al servicio de México"

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SUBDIRECCIÓN ACADÉMICA

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO

C. c. p. Departamento de Ciencias Computacionales
Departamento de Servicios Escolares

CMAZ/LMZ

EDUCACIÓN
10 AGO 2023
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SERVICIOS ESCOLARES
RECIBIDO

Cuernavaca, Mor., 26/junio/2023

OFICIO No. DCC/153/2023
Asunto: Aceptación de documento de tesis
CENIDET-AC-004-M14-OFICIO

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial de CÉSAR DAVID REY FIGUEROA con número de control M21CE023, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis de grado titulado "MEJORA DEL ALGORITMO FUZZY C-MEANS MEDIANTE EL PARADIGMA DE PROGRAMACIÓN DISTRIBUIDA Y/O PARALELA"y hemos encontrado que se han atendido todas las observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

JOAQUÍN PÉREZ ORTEGA
Director de tesis

MARÍA YASMÍN HERNÁNDEZ PÉREZ
Codirectora de tesis

ALICIA MARTÍNEZ REBOLLAR
Revisor 1

JOSÉ CRISPÍN ZAVALA DÍAZ
Revisor 2

JAVIER ORTIZ HERNÁNDEZ
Revisor 3

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
30 JUN 2023
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SERVICIOS ESCOLARES
RECIBIDO

C.c.p. Depto. Servicios Escolares.
Expediente / Estudiante

Dedicatoria

A mi esposa Emily Salgado por todo el apoyo y el amor que me ha dado y por motivarme a cumplir mis objetivos.

A mis padres por preocuparse por mi bienestar y por todo su apoyo, en especial a mi madre por todo su cariño y comprensión.

A mis hermanos por su orientación y apoyo en algunos de los obstáculos que se me han presentado en la vida.

Agradecimientos

Agradezco a Dios por la vida, por la salud, por las personas que puso en mi camino y la capacidad que me dio para lograr mis metas.

Agradezco a CONAHCYT por el apoyo económico para desarrollar esta tesis.

Agradezco al CENIDET y al TecNM por brindarme la oportunidad de desarrollar la Maestría en Ciencias de la Computación.

Agradezco al Dr. Joaquín Pérez Ortega, a la Dra. Leticia Sánchez Lima y a la Dra. Sandra Silvia Roblero Aguilar por la orientación y el apoyo que me han brindado para desarrollar mi tema de tesis. Agradezco a mi codirectora de tesis la Dra. María Yasmin Hernández Pérez y a los miembros del comité tutorial el Dr. José Crispín Zavala Díaz, la Dra. Alicia Martínez Rebollar y el Dr. Javier Ortiz Hernández, por ayudarme con sus observaciones y comentarios, a elaborar este documento. De igual manera agradezco a los miembros del grupo de seminario de investigación y a todos los profesores que de algún modo me brindaron conocimiento necesario para desarrollar mi tesis.

Resumen

La investigación realizada, se ubica en el contexto de los problemas de agrupamiento. El problema específico que se abordó, consistió en reducir el tiempo de procesamiento del algoritmo de agrupamiento Fuzzy C-Means (FCM), mediante programación paralela.

El problema de agrupamiento al cual busca dar solución FCM, es del tipo *NP-Hard*. Por esta razón se justifica el uso de métodos heurísticos para su solución. En este sentido, se han realizado diferentes investigaciones que implementaron un enfoque paralelo en el algoritmo FCM. Sin embargo, la mayoría de dichas investigaciones proponen algoritmos enfocados en un dominio específico. En contraste, el algoritmo propuesto en esta investigación, denominado *Parallel Optimization Fuzzy C-Means* (POFCM), es de propósito general, orientado a resolver grandes *datasets* y está inspirado en una variante secuencial de FCM altamente eficiente, denominada *Hybrid OK-Means Fuzzy C-Means* (HOFCM).

Para realizar la implementación paralela, se utilizó la técnica *OpenMP*, con la cual se obtiene un algoritmo paralelo escalable, portable y adaptable. Con esta implementación se utilizaron todos los procesadores disponibles en un equipo de cómputo.

Para validar los resultados se diseñaron y ejecutaron diversos experimentos. Los resultados que mostró la implementación de POFCM fueron excelentes. En el mejor de los casos al ser comparado con FCM estándar secuencial, se redujo hasta en un 94.98% el tiempo de ejecución. Por otra parte, al comparar POFCM con HOFCM, en el mejor de los casos se obtuvo una eficiencia paralela superior a 0.9.

Es destacable que POFCM puede ser útil para usuarios que busquen ejecutar grandes *datasets* en un tiempo razonable, incluso en un equipo de cómputo convencional.

Abstract

The research performed in the context of clustering problems. The specific problem that was addressed consisted of reducing the processing time of the Fuzzy C-Means (FCM) clustering algorithm, through parallel programming.

The clustering problem FCM seeks to solve is of the NP-Hard type, for which the use of heuristic methods for its solution is justified. In this regard, several works have implemented a parallel approach in FCM. However, most of those proposed algorithms are oriented to a specific domain. In contrast, the algorithm proposed in this paper is general purpose. It is inspired by a highly efficient FCM variant called Hybrid OK-Means Fuzzy C-Means (HOFCM), focused on solving large datasets. This proposed algorithm was called Parallel Optimization Fuzzy C-Means (POFCM).

The OpenMP technique was used for the parallel implementation, with which a scalable, portable, and adaptable parallel algorithm is obtained. With this parallel implementation, all available processors in a computer are used.

The implementation of POFCM obtained excellent experimental results; compared with standard sequential FCM, the execution time was reduced by up to 94.98%. In addition, when comparing HOFCM with POFCM in the best cases, values for parallel efficiency greater than 0.9 were obtained when running real and synthetic datasets. Additionally, POFCM was tested with large datasets such as those presented in the big data to observe its behavior; in these tests, the execution time of HOFCM was reduced by up to 66.6%.

Finally, POFCM can be useful for users looking to run large datasets in a reasonable amount of time, even on a conventional computer.

CONTENIDO

Página

Lista de tablas

Página

Lista de figuras

Página

Capítulo 1

INTRODUCCIÓN

En este capítulo, se exponen las siguientes secciones: Sección 1.1, se presentan los argumentos que motivaron esta investigación; en la Sección 1.2, se expone brevemente el contexto del agrupamiento de datos; en la Sección 1.3, se plantea el problema en el cual se enfoca esta investigación; en la Sección 1.4, se exponen las razones que justifican la presente investigación; en la Sección 1.5, se presenta el método con el cual se dio solución al problema planteado; en la Sección 1.6, se presentan los objetivos que orientaron esta investigación; en la Sección 1.7, se exponen los alcances y las limitaciones que se presentaron al realizar este trabajo; y por último, en la Sección 1.7, se explica la organización del presente documento.

1.1 Motivación

En los últimos años, los avances tecnológicos en todos los campos de conocimiento, han generado un aumento exponencial en la producción y almacenamiento de información. Una manera de utilizar esa información valiosa, a partir las grandes cantidades de datos, es mediante el agrupamiento. El proceso de agrupamiento es un método por el cual los datos se dividen en grupos, de tal forma que los objetos de cada grupo compartan características similares entre sí, y a la vez estas características son diferentes a las de otros grupos [1], [2]. El agrupamiento de datos es utilizado en diversos campos del conocimiento computacional, tales como como Ciencia de Datos, Minería de Datos [3], Reconocimiento de Patrones y Procesamiento de Imágenes [4], entre otras.

El problema que se aborda en esta investigación, es materia de gran interés para la comunidad científica por su vigencia y actualidad. En el CENIDET, desde hace varios años se han desarrollado investigaciones y tesis relacionadas con los algoritmos de agrupamiento, entre ellos se destacan Fuzzy C-Means (FCM) y K-Means. Se ha puesto mayor atención a estos algoritmos de agrupamiento debido los buenos resultados que presentan en tiempo y calidad.

1.2 Contexto de la investigación

El agrupamiento de datos, es una técnica de aprendizaje no supervisado, el cual está siendo utilizado en diversas áreas como la Inteligencia Artificial y la Minería de Datos. Esta técnica, ha cobrado mayor importancia conforme aumenta la cantidad de datos que se generan a nivel mundial.

El agrupamiento de datos, no tiene un origen preciso, por esta razón, no se puede identificar quien fue su creador. Ha sido un concepto en constante evolución, el cual, por su importancia, se ha desarrollado a lo largo de varios años. No obstante, se podría mencionar como uno de los pioneros en introducir el término de agrupamiento a Joseph Zubin [5], quien en 1938 comenzó a aplicar técnicas de análisis de grupos en la psicología.

El agrupamiento de datos se usa en una amplia gama de aplicaciones, por ejemplo, Ciencia de Datos, Minería de Datos [6], [7] y Segmentación de Imágenes [8], por mencionar algunas. Para desarrollar el agrupamiento de datos, se aplican distintos algoritmos de agrupamiento. Con dichos algoritmos de agrupamiento se pueden identificar diversos patrones en los datos. Un análisis detallado de estos patrones puede apoyar a las empresas, científicos o gobiernos para la toma de mejores decisiones.

Tradicionalmente, los algoritmos de agrupamiento se han dividido en jerárquicos y particionales [9]. Los primeros, producen una serie anidada de particiones de los datos. Algunos ejemplos son los algoritmos *Single-Link, Complete-link y Mínimum Variance*. Los algoritmos particionales, a su vez, se dividen en duros y difusos. Estos algoritmos, producen una partición de los datos,

algunos ejemplos son los algoritmos K-Means y Fuzzy C-Means (FCM) [10], estos son los dos algoritmos particionales más conocidos.

Esta investigación, se centra en el algoritmo de agrupamiento FCM, el cual es de cierto modo similar a K-Means, la diferencia entre estos dos algoritmos de agrupamiento radica, principalmente, en que K-Means es un algoritmo de agrupamiento duro y FCM es un algoritmo de agrupamiento difuso.

El termino de K-Means fue utilizado por primera vez por MacQueen en 1967 [11]. No obstante, Lloyd en 1957 propuso el algoritmo estándar en un marco de investigación sobre técnicas de cuantificación vectorial. Pero la idea original se remonta a Steinhaus en 1956.

K-Means es un algoritmo de agrupamiento duro, en donde los objetos tienen una pertenencia de 1 o 0 con respecto a los grupos. El proceso de K-Means consiste en los siguientes cuatro pasos:

1. Inicialización: se eligen *k* objetos al azar como centroides iniciales. Donde *k* es el número de grupos.
2. Clasificación: se asigna cada objeto a su centroide más cercano.
3. Cálculo de centroides: se calcula la media de los objetos en cada grupo y se utiliza como nuevo centroide.
4. Criterio de paro: se repiten los pasos 2 y 3 hasta que no hay cambios significativos en los grupos o se alcana un número máximo de iteraciones establecido.

Por otro lado, FCM tiene su origen en la lógica difusa. Uno de los pioneros de la lógica difusa es Lotfi Zadeh [12] en 1965. En 1969, Ruspini publicó un artículo que se convirtió en la base para la mayoría de algoritmos difusos [13]. Sus ideas establecieron la estructura para la partición difusa, además, describieron y ejemplificaron el primer algoritmo para lograrla. Posteriormente, FCM fue propuesto por Dunn [14] y generalizado por Bezdek [15].

El algoritmo FCM particiona los conjuntos de datos en subconjuntos, por medio de la creación de centroides. A diferencia del algoritmo duro, este no

particiona los objetos de forma binaria. Al ser un algoritmo difuso, cada objeto puede pertenecer a más de un grupo y también puede tener distintos grados de pertenencia, con respecto a distintos centroides [16].

En la mayoría de los casos, FCM utiliza como medida de similitud la distancia euclidiana, la cual supone que cada característica tiene la misma importancia para FCM. En la mayoría de los problemas reales, se presenta una mayor o menor importancia para algunas características. Para FCM, el agrupamiento de datos, se logra mediante un proceso de optimización iterativo que minimiza una función objetivo [9].

1.3 Descripción del problema

Se sabe que el problema al cual busca dar solución el algoritmo FCM es un problema *NP-Hard* [17], lo cual justifica la necesidad de utilizar métodos heurísticos para lograr la solución de grandes instancias de datos en un tiempo razonable.

La complejidad temporal de FCM en cada iteración es nc^2d, en donde n es el número de objetos, c los grupos y d las características o atributos de los datos. En la Figura 1, se presenta una gráfica con propósitos ilustrativos, en la cual se muestra cómo

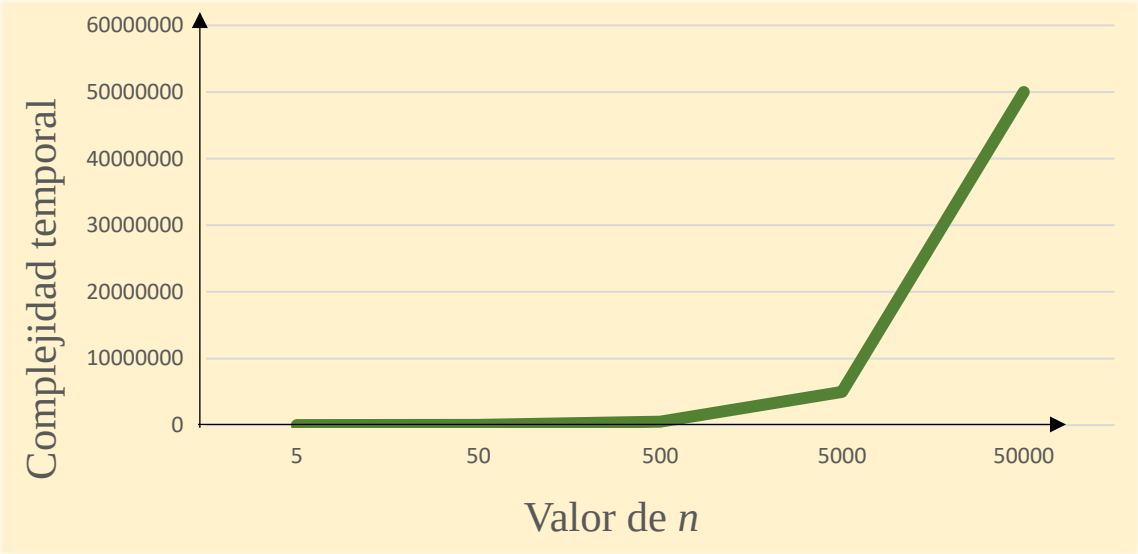

Figura 1. Complejidad temporal de FCM

aumenta la complejidad temporal de FCM de forma exponencial, al incrementar el tamaño de la instancia a resolver.

En este sentido, aunque FCM presenta una solución para el agrupamiento de datos de forma difusa, su tiempo de ejecución es considerablemente mayor en comparación con el de otros algoritmos de agrupamiento como es el caso de K-Means. Por esta razón resolver grandes instancias de datos con FCM en un tiempo razonable, actualmente, es un problema que debe resolverse.

Para dar solución a este problema, se han desarrollado diversas mejoras para el algoritmo FCM como [18] [19], algunas de estas mejoras, buscan reducir el tiempo de ejecución del algoritmo para grandes instancias de datos con modificaciones en la inicialización, la clasificación o en el criterio de paro. No obstante, a pesar de que dichas mejoras vuelven más eficiente al algoritmo, suelen ser secuenciales, por lo que no reducen el tiempo de ejecución de la manera que lo haría su versión paralela, al utilizar todos los procesadores disponibles.

1.4 Justificación

Con el desarrollo de nuevas tecnologías, se ha incrementado el número de procesadores disponibles en las computadoras. Sin embargo, los programas secuenciales siguen siendo comunes y únicamente utilizan un procesador a la vez, desaprovechando la capacidad de un equipo de cómputo con varios procesadores.

Como ya se ha mencionado, ha aumentado de manera exponencial la cantidad de datos que se generan, por lo tanto, es necesario resolver grandes instancias en poco tiempo.

En este sentido se ha comprobado en diversos trabajos que paralelizar un algoritmo puede ser una alternativa eficiente para reducir el tiempo de ejecución de un algoritmo. En esta investigación, se propone paralelizar una variante de FCM denominada HOFCM, con el objetivo de disminuir el tiempo de ejecución de esta variante, aproximadamente, en 4 veces al utilizar 4 procesadores.

1.5 Método de solución

Para la presente investigación, se propuso paralelizar una variante secuencial del algoritmo FCM con el objetivo de disminuir su tiempo de ejecución.

Con este propósito, se desarrolló un algoritmo paralelizado con la técnica *OpenMP*, denominado *Parallel Optimization Fuzzy C-Means* (POFCM) basado en la variante secuencial denominada HOFCM [19]. Esta es una eficiente variante del algoritmo FCM enfocada en la solución de grandes instancias de datos. La solución paralela desarrollada en esta investigación, se comparó experimentalmente, con los algoritmos secuenciales HOFCM y FCM.

Para la implementación del algoritmo se aplicó programación paralela con el tipo de arquitectura maestro-esclavo mediante *OpenMP* el cual tiene una estructura de *fork-join*. Para paralelizar la variante seleccionada Se realizaron los siguientes pasos:

- Se seleccionó una variante secuencial del algoritmo FCM, orientada a reducir el número de iteraciones al resolver grandes instancias.
- Se implementó dicha variante, así como el algoritmo FCM estándar.
- Se mejoró la variante seleccionada mediante el paradigma de programación paralela.
- Se realizaron pruebas con *datasets* reales y sintéticas de gran tamaño. los resultados de la implementación paralela se validaron de manera experimental.
- Se compararon, de manera experimental, los resultados de la implementación paralela, con los resultados que se presentan en otras publicaciones.

1.6 Objetivos

Objetivo general:

Reducir el tiempo de procesamiento para una variante secuencial del algoritmo de agrupamiento Fuzzy C-Means, por medio de una mejora paralela.

Objetivos específicos:

a) Seleccionar una variante secuencial de FCM que sea eficiente al resolver grandes instancias.

b) Mejorar, mediante el paradigma de programación paralela, la variante seleccionada.

c) Validar los resultados de la mejora propuesta, mediante la solución de instancias de prueba sintéticas y reales.

d) Cuantificar la mejora propuesta, mediante métodos experimentales.

1.7 Alcances y limitaciones

Alcances:

a) La mejora propuesta se comparará con respecto a los resultados presentados en los documentos examinados y algoritmos implementados.
b) La mejora propuesta priorizará reducir el tiempo de ejecución del algoritmo al resolver grandes instancias.
c) Los datos de prueba reales serán obtenidos de los repositorios del UCI.

Limitaciones:

a) La validación de los resultados será de manera experimental.
b) Las pruebas únicamente se realizarán con equipo disponible dentro del Cenidet o al cual se tenga acceso.

1.8 Organización del documento

A continuación, se presenta la estructura del presente documento: En el Capítulo 2, se desarrolla el estado del arte en el cual se presentan las investigaciones más relevantes con respecto al problema de esta investigación; en el Capítulo 3, se expone el motivo de la selección de la variante secuencial de FCM y se detalla la estructura de dicha variante; en el Capítulo 4, se explica cómo se desarrolló la paralelización de la mejora propuesta, denominada POFCM; en el Capítulo 5, se describe el diseño de los experimentos realizados para hacer su validación; en el Capítulo 6, se muestran los resultados experimentales obtenidos con la implementación de POFCM; por último, en el Capítulo 7, se presentan las conclusiones y se recomiendan temas para trabajos futuros.

Capítulo 2

TRABAJOS RELACIONADOS

En este capítulo, en la sección 2.1, se presenta una breve síntesis acerca del origen de Fuzzy C-Means (FCM); En la Sección 2.2, se muestran algunas de las mejoras secuenciales que se han implementado en FCM; En la Sección 2.3, se muestran los trabajos paralelos más cercanos, principalmente, los que utilizan *OpenMP*; Por último, en la Sección 2.4, se presentan otras investigaciones relacionadas con la paralelización de FCM y sus variantes.

2.1 Origen de FCM

En esta Sección se presenta un breve texto acerca de los orígenes de FCM, comenzando por el concepto de agrupamiento difuso y posteriormente, mencionando a los principales contribuyentes para la creación de FCM.

2.1.1 Concepto de agrupamiento difuso

El concepto de los conjuntos difusos se presentó en 1965 en el artículo “Fuzzy sets” [20] publicado por Lotfi Zadeh. En este artículo se presentan como una extensión de la teoría de los conjuntos clásicos, debido a que se observó que, en la mayoría de las aplicaciones reales, en las cuales se modelan objetos o conceptos, éstos no tienen una pertenencia clara a un único conjunto.

2.1.2 Origen del algoritmo difuso

En 1969, Ruspini publicó un artículo que se convirtió en la base para la mayoría de algoritmos difusos [13]. Ruspini presentó el enfoque de clasificación basado en la teoría difusa. Sus ideas establecieron la estructura para la partición difusa y describieron y ejemplificaron el primer algoritmo para lograrla. Ruspini fue el primero en plantear el agrupamiento difuso como un problema de optimización, basado en una función objetivo.

El algoritmo FCM fue propuesto por Dunn y generalizado por Bezdek [21], [22]. Para este algoritmo, la agrupación de datos se logra mediante un proceso de optimización iterativo que minimiza una función objetivo. El problema básico del agrupamiento de datos es separar un conjunto de objetos en grupos similares entre sí [23].

Como se mencionó anteriormente, el algoritmo FCM tiene su origen en la lógica difusa. Dunn en 1973, en el artículo "*A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters*" [14], propone la base para el desarrollo del algoritmo FCM la cual es Fuzzy ISODATA. Utiliza lógica difusa para asignar los objetos a los conjuntos. A diferencia de los métodos de agrupamiento duros que asignan a cada objeto un solo grupo, la técnica de agrupamiento difuso permite que cada objeto tenga un cierto grado de pertenencia con respecto a diversos conjuntos.

2.1.3 Algoritmo FCM

Bezdek desarrolló FCM como se le conoce hoy en día, con base en los avances previos de la lógica difusa, uno de los artículos que mejor detalla el algoritmo FCM es "*FCM: The Fuzzy C-Means Clustering Algorithm*" publicado en 1984 [15]. FCM es un algoritmo que se inspira en el algoritmo K-Means, pero a diferencia de KM, FCM utiliza la lógica difusa por lo que se asignan grados de pertenencia a los objetos con respecto a los conjuntos. A diferencia de K-Means en donde el agrupamiento es duro y cada objeto pertenece a un conjunto único.

En el artículo mencionado también se señala que FCM es un algoritmo iterativo que requiere de ciertos parámetros de entrada, como lo son, el número de grupos. En este artículo se muestra la implementación del algoritmo FCM en Fortran IV para un determinado conjunto de datos, en este artículo para realizar el agrupamiento difuso, se estableció el factor como m=2 y se utilizó un criterio de convergencia con valor de épsilon=0.01, el cual es un umbral que señala que el algoritmo debe detenerse cuando el valor absoluto de la máxima diferencia de los grados de pertenencia de un objeto en dos ejecuciones consecutivas sea menor al umbral previamente establecido.

2.2 Variantes del algoritmo FCM

Existen diversas variantes para mejorar alguna de las fases del algoritmo FCM. Algunas de estas mejoras se desarrollaron con el propósito de solucionar un problema específico, en cierta área de aplicación. De esta manera se busca resolver necesidades de distintos sectores como el de la salud, la agricultura y la tecnología, entre otros.

Existen mejoras a FCM que tienen un enfoque general y buscan mejorar alguna fase del algoritmo con el fin de: reducir el tiempo de ejecución del algoritmo, mejorar la calidad de la solución o ambas.

Para el algoritmo FCM se definen cuatro fases principales, las cuales son: inicialización, clasificación, recálculo de centroides y criterio de paro. A continuación, se presentarán algunas mejoras enfocadas en la reducción de tiempo de solución del algoritmo.

2.2.1 FCM ++

Uno de los artículos más conocidos acerca de mejoras de FCM en la fase de inicialización es el artículo denominado "*Fuzzy C-means++: Fuzzy C-means with effective seeding initialization*" [18], en el cual se propone una mejora hibrida para la fase de inicialización, debido a que esta fase tiene un gran impacto en el tiempo de solución y la calidad final del agrupamiento, dependiendo del lugar

inicial dónde se localicen los centroides, el algoritmo podría tener mejores o peores resultados.

La mejora propuesta en este artículo, utiliza un método que consiste en elegir centroides iniciales de manera iterativa con K++ [24], seleccionando mediante una probabilidad nuevos centroides, los cuales se encuentren alejados de los que fueron previamente seleccionados. En este caso el algoritmo FCM se inicializa con centroides y no con la matriz de pertenencia.

Los resultados experimentales muestran que se obtienen mejores resultados al ejecutar la variante propuesta, en comparación con el método tradicional de semilla aleatoria. Se ejecutó con tres *datasets* reales, para uno de estos *datasets* se logró que la ejecución fuese hasta 1.44 veces más rápido. También se destaca que FCM ++ es capaz de manejar con mayor efectividad, distribuciones de datos con formas no esféricas y asimétricas.

2.2.2 HOFCM

El artículo más relevante para llevar a cabo el propósito de esta investigación es "*Hybrid Fuzzy C-Means Clustering Algorithm Oriented to Big Data Realms*" [19], el cual propone una mejora en la fase de inicialización de FCM con base en los algoritmos K++ [24] y OK-Means [25]. Se considera relevante debido a los siguientes criterios:

a) Logra una reducción de tiempo significativa en la mayoría de los casos.
b) Mejora la calidad del agrupamiento en la mayoría de los casos.
c) Tiene un enfoque general y no de un área específica.

La reducción de tiempo se logra al seleccionar de forma eficiente la matriz de pertenencia para FCM. Esto se logra al realizar 10 ejecuciones de K++ y OK-Means. Al final de esas ejecuciones, se seleccionan los centroides en cuya ejecución se obtuvo el menor valor de la función objetivo. Posteriormente, con esos centroides se calcula la matriz de pertenencia la cual se proporciona al algoritmo FCM, con eso se reduce el número de iteraciones y, por lo tanto, el

tiempo de ejecución. Adicionalmente, en la mayoría de los casos se mejora la calidad de la solución, sobre todo para *datasets* con varias dimensiones.

Esta mejora se ejecutó con cuatro *datasets* reales y tres sintéticos. En el mejor de los casos, se obtuvo una reducción de tiempo con respecto a FCM estándar de 93.94%.

2.3 Trabajos paralelos más cercanos

En esta Sección se presenta una recopilación de las investigaciones, que se han encontrado relevantes, con respecto a mejorar de manera paralela el algoritmo FCM.

La investigación presentada en este documento, específicamente, utiliza *OpenMP* para su implementación. Este método de paralelización utiliza memoria compartida mediante la estructura *fork-join* para realizar las tareas de forma paralela. *OpenMP* es una alternativa sencilla para crear programas paralelos debido a que su sintaxis es de fácil comprensión

A pesar de que en la literatura existen diversas investigaciones en torno a K-Means y sus variantes, paralelizadas con *OpenMP* se detectaron pocos artículos en los cuales se mencione FCM paralelizado con *OpenMP*. A continuación, se presentan las investigaciones más cercanas.

2.3.1 FCM paralelo en *OpenMP*

Una de las investigaciones en las cuales se paraleliza FCM con *OpenMP* es "*Applying Fuzzy clustering method to color image segmentation*" publicada en el 2015 [26]. Los autores implementan una mejora paralela en FCM utilizando *OpenMP* con enfoque en la segmentación de imágenes. Se destaca cómo se utiliza FCM con el fin de lograr la segmentación de imágenes a color donde se agrupan los píxeles de una imagen en distintos grupos con un cierto grado de pertenencia.

Se resalta en el algoritmo que, debido al gran coste computacional del algoritmo, el cual puede tener un tiempo de ejecución elevado, se propone una

implementación paralela para asignar los distintos píxeles a los grupos. Los resultados experimentales muestran que al utilizar *OpenMP* para paralelizar FCM en particular se reduce significativamente el tiempo de procesamiento. En promedio se obtuvo que para 10 ejecuciones con 16 grupos se redujo, aproximadamente, hasta en un 50%. Además, es importante destacar que la mejora paralela no afecta la calidad de la segmentación obtenida, por lo que paralelizar los algoritmos con *OpenMP* se convierte en una técnica atractiva para la segmentación de imágenes.

2.3.2 Hesitant FCM paralelo en *OpenMP*

El artículo "*Parallel hesitant fuzzy C-means algorithm to image segmentation*" [27] publicado en el año 2022, se investiga un método basado en la teoría de conjuntos difusos *Hesitant* y se utiliza un algoritmo denominado Hesitant Fuzzy C-Means (HFCM) para mejorar la segmentación de imágenes. Los experimentos reportados en este artículo, mostraron una mejora en la asignación de los píxeles en los bordes de las regiones, lo cual proporcionó regiones más homogéneas.

Se señala que, en comparación con otros métodos de la literatura, con el método propuesto se obtuvo, un mejor rendimiento en métricas como *accuracy*. Una contribución adicional de este artículo, es la implementación de *OpenMP* para paralelizar el código mediante bucles, lo que reduce considerablemente el tiempo y permite que el algoritmo converja en menos iteraciones que su contraparte secuencial. La paralelización se divide en dos fases principales, en las cuales se paraleliza la matriz de incertidumbre difusa y la actualización de las variables del algoritmo HFCM. la paralelización se hace a nivel de bucle, por lo que se pueden ejecutar iteraciones de forma independiente. En el artículo se muestran los tiempos secuenciales y paralelos, en los cuales se observa una reducción del tiempo de ejecución mayor al 70% para el mejor resultado.

2.4 Otros trabajos paralelos de interés

En la literatura existen diversas investigaciones paralelas acerca de FCM. La mayoría de las investigaciones, desarrolladas recientemente, utilizan la Unidad de Procesamiento Grafico (GPU) la cual se refiere al microprocesador diseñado específicamente para procesar y optimizar gráficos de computadora, debido a sus excelentes resultados para reducir tiempo. En este apartado se exponen algunas investigaciones que se consideraron de interés debido a sus resultados o métodos empleados, relacionados con el algoritmo FCM.

Se encontró es una recopilación de artículos paralelos acerca de FCM denominada "*A Survey and Systematic Categorization of Parallel K-means and Fuzzy-c- Means Algorithms*" del año 2019 [28]. En esta recopilación, presenta una investigación acerca de las implementaciones paralelas de los algoritmos de agrupamiento K-means y Fuzzy C-Means, en entornos paralelos como: *Hadoop*, *MapReduce*, *Spark* o en unidades de procesamiento gráfico y sistemas de procesamiento multinúcleo. En este artículo, se menciona que se revisaron aproximadamente, 92 artículos, publicados entre el año 2000 y 2018. Con respecto a la investigación de este documento, se encontró relevante la clasificación que tiene de los artículos paralelos y algunos de esos artículos se añadieron a este documento.

2.4.1 FCM paralelizado con MPI

Existen distintas formas de paralelizar FCM, por lo cual se pueden utilizar distintos métodos. La primera forma que aquí se presenta, es por medio de la interfaz de paso de mensaje (MPI), la cual consiste en una biblioteca de comunicación utilizada para programar aplicaciones en paralelo. Algunas de las principales ventajas de paralelizar con MPI son:

1. Es una biblioteca de bajo nivel, lo que brinda al programador un alto control sobre la forma en que se comunica y maneja la memoria del equipo. Esto permite una eficiente implementación de paralelismo en los algoritmos.

2. Es altamente escalable, por lo que puede manejar un gran número de procesos con distintos nodos sin reducir el rendimiento, esto lo vuelve altamente recomendable para procesar grandes volúmenes de datos.
3. Es una biblioteca de comunicación de propósito general, por lo que se puede utilizar en una amplia variedad de plataformas y arquitecturas de hardware. Esto proporciona la facilidad de ejecutar en diferentes equipos el mismo algoritmo.

A pesar de las ventajas que se tienen al paralelizar con MPI, también tiene algunas desventajas, como las siguientes:

1. Utiliza una sintaxis más compleja que otros métodos para paralelizar, ya que el programador gestiona de forma explícita la comunicación entre los distintos nodos.
2. Tiene un alto consumo de recursos debido a que comunica distintos procesos y espacios en memoria creados para ejecutarse en un entorno de memoria distribuida.
3. No es adecuado en todos los problemas debido a que en algunas situaciones el costo de comunicación entre nodos puede ocasionar una disminución en el rendimiento.

Una de las primeras implementaciones paralelas de FCM, fue propuesta en el año 2002, en el artículo "*Parallel Fuzzy c-Means Clustering for Large Data Sets.*" [9]. Los autores implementaron de forma paralela el algoritmo FCM mediante la interfaz de paso de mensaje (MPI), utilizando un servidor denominado *AlphaServer*, el cual contaba con 128 procesadores. En este caso, se divide el conjunto de datos en varias partes y utiliza los nodos de procesamiento para realizar el cálculo de la clasificación.

Las pruebas realizadas compararon la aceleración obtenida, para distintas cantidades de procesadores con el algoritmo PKM. Se señala que se obtuvo una aceleración casi ideal para datasets de gran tamaño. Además, los resultados mostraron que la implementación paralela fue eficiente al reducir el tiempo de procesamiento sin disminuir la calidad del agrupamiento.

2.4.2 FCM paralelizado basado en la nube

Uno de los métodos utilizados comúnmente para implementar algoritmos paralelos con memoria distribuida es el cómputo distribuido basado en la nube. Este método permite conectar múltiples equipos de cómputo a través de internet, ya sea por servicios de nube pública o privada. Los recursos computacionales están interconectados y disponibles bajo demanda. Algunas de las ventajas principales de este tipo de programación paralela son las que se enlistan a continuación:

1. Permite modificar los recursos de manera simple, ya sea añadiendo más recursos o eliminándolos, lo que permite escalar un sistema de forma simple.
2. Permite acceder de forma remota a los recursos, lo que posibilita utilizarlos desde distintos lugares.

Por otro lado, algunas de las desventajas que implica el utilizar cómputo en la nube son las siguientes:

1. Tiene dependencia a la red, lo que genera que gran parte del rendimiento se base en la calidad y disponibilidad de la conexión.
2. Puede estar expuesto a problemas de seguridad si no se implementan las medidas necesarias para proteger la conexión.
3. Tienen un control externo, debido a las políticas y decisiones de los proveedores de la nube, lo que limita el control sobre el sistema.

En el artículo denominado "*Distributed Fuzzy C-Means algorithms for big sensor data based on cloud computing*" [29], del año 2015. Se presenta un enfoque paralelo distribuido en la nube. Se proponen tres variantes basadas en FCM: DFCM, DOFCM y DKFCM. En DFCM se distribuyó FCM; En DOFCM se distribuyó FCM *online*; Por último, en DKFCM se distribuyó Kernel FCM. Los experimentos realizados en conjuntos de datos de sensores reales muestran que los algoritmos paralelos propuestos logran una reducción de tiempo significativa

y poseen escalabilidad. Se encontró que DOFCM obtuvo los mejores resultados para dos *datasets* de gran tamaño.

Otra investigación que utilizó cómputo en la nube fue publicada en el artículo "*Distributed k-Means Algorithm and Fuzzy c-Means Algorithm for Sensor Networks Based on Multiagent Consensus Theory*" en el año 2017 [30]. Se proponen dos algoritmos distribuidos, uno basado en K-Means y otro en Fuzzy C-Means, ambos para redes de sensores inalámbricos. Adicionalmente, se propone un algoritmo de K ++ distribuido para la selección de centroides.

Los experimentos realizados muestran una reducción de tiempo para ambas implementaciones. Para FCM se prueba el algoritmo para 3,600 observaciones, para k= 6, 9 y 12. El mejor resultado se obtuvo para k= 9, con el cual se logró disminuir el tiempo de ejecución en aproximadamente 43%. Se enfatiza que el algoritmo distribuido basado en FCM tiene escalabilidad.

2.4.3 Cómputo paralelo hibrido utilizando GPU

El implementar algoritmos híbridos GPU-CPU se ha convertido en una alternativa viable, aunque poco accesible debido a los altos costos del GPU. Paralelizar programas de manera híbrida ha mostrado que es posible reducir considerablemente el tiempo de ejecución al resolver grandes instancias.

En el artículo "*A GPU-based implementation of the fuzzy C –means*" en el año 2015[31]. Se propone una versión paralela del algoritmo brFCM implementada en CPU-GPU, para la segmentación de imágenes médicas. Se realizan los experimentos utilizando imágenes médicas. Se comparan la mejora paralela en GPU con el algoritmo brFCM secuencial implementado en CPU y se obtiene que la mejora propuesta es 2.24 veces más rápida.

En el año 2017 en "*New Parallel Hybrid Implementation of Bias Correction Fuzzy C-means Algorithm*" [32]. Se propone un algoritmo basado en FCM para imágenes de resonancia magnética del cerebro, denominado *Bias Correction Fuzzy C-Means* (BCFCM). Se proponen dos versiones paralelas para este

algoritmo y se utiliza GPU para las ejecuciones. Se obtiene hasta una aceleración de 24 veces con respecto al secuencial.

En el año 2018 en el artículo "*Improving fuzzy C-mean-based community detection in social networks using dynamic parallelism*" [33]. Se presentan tres implementaciones del algoritmo FCM base, que utilizan GPU. La primera (HCG) es híbrida y utiliza GPU y CPU; La segunda (DP) utiliza paralelismo dinámico; La tercera (HNP) es un algoritmo anidado paralelo híbrido. Se utilizan dos *datasets* para realizar las pruebas, se compara la aceleración de los algoritmos mejorados y se señala que la mejor implementación es HNP con una aceleración de 12.58, seguido de HCG con una aceleración de 8.29 y por último DP con 4.45.

En el año 2020 en "*Parallel Implementation for 3D Medical Volume Fuzzy Segmentation*" [34]. Se propone una versión modificada de FCM para la segmentación de imágenes 3D. Se presenta una implementación paralela del algoritmo propuesto utilizando GPU. Se utilizan *datasets* reales para validar el algoritmo. Se obtiene hasta un rendimiento de aproximadamente, 23 veces más rápido que FCM base.

En el año 2020 en el artículo "*Knowledge based fuzzy c-means method for rapid brain tissues segmentation of magnetic resonance imaging scans with CUDA enabled GPU machine*" [35]. Se propone un método llamado FCM-GENIUS el cual busca acelerar la obtención de conocimiento de imágenes de resonancia magnética de la cabeza humana, para la segmentación de imágenes del tejido cerebral. Consta de tres pasos y utiliza a FCM para la optimización. Se compara el método propuesto con FCM y se obtiene que logra hasta una aceleración de 7 veces utilizando el paralelismo en GPU.

2.5 Comparación de la propuesta paralela

Como se mostró anteriormente, FCM ha sido paralelizado utilizando distintas plataformas y arquitecturas, como lo son: interfaz de paso de mensaje (MPI); cómputo en la nube; GPU; Spark; Map Reduce; OpenCL y *OpenMP*. Para

contrastar el trabajo propuesto con algunos de los trabajos más relevantes, encontrados en la literatura, se realizó la Tabla 1, la cual proporciona información acerca de las variantes paralelas de FCM que se han implementado. Tiene la siguiente distribución: la primera Columna muestra el algoritmo propuesto e incluye la referencia del artículo, en el cual fue propuesto; en la segunda Columna se indica si la implementación es paralela; la tercera Columna señala si la implementación es distribuida; la cuarta Columna menciona la plataforma o arquitectura en la cual fue implementado el algoritmo; la quinta Columna indica si la propuesta de mejora fue en la fase de inicialización; la sexta Columna indica si el algoritmo es híbrido; la séptima Columna menciona las características principales del equipo utilizado para la experimentación; la octava Columna describe el *dataset* más grande reportado, en general el número de objetos se menciona primero y posteriormente, después de la x, se menciona el número de características, sin embargo, algunos autores expresan el tamaño del *dataset* por medio del tamaño del archivo en bytes o en píxeles, en estos casos se señalan las unidades utilizadas; en la novena Columna se menciona el número de *dataset*s utilizadas seguido de una *R* si el *dataset* es real o una *S* si el *dataset* es sintético. Las últimas tres columnas, indican cómo el autor de cada artículo compara sus resultados; la décima Columna indica si el trabajo fue comparado con FCM estándar secuencial; la onceava Columna indica si la comparación se realizó con algoritmos propuestos por otros autores; por último, la doceava Columna indica si se comparó la versión paralela propuesta, con su versión secuencial.

Con base en la Tabla 1 se pueden señalar los siguientes puntos relevantes:

a) Hay pocos algoritmos basados en FCM, que utilicen la técnica de *OpenMP* para paralelizar.
b) Se observa que tanto los procesadores i5 como los i7, son los predominantes en las pruebas, estos procesadores son comunes en equipos de cómputo personales.
c) La mayoría de trabajos utilizan menos de 6 datasets.

Tabla 1. Comparación entre algoritmos paralelos basados en FCM

Algoritmo	P	D	Plataforma	I	H	Características del equipo	Tamaño (n x d)	Tipo	Sec. FCM	Otros	Variante sec.
DFCM DOFCM DKFCM [29]		✓	Cloud base Map Reduce	✓	✓	20 máquinas, Intel® Core™ i7 3.2 GHz, 8 GB RAM. 2 TB disco duro	(30,000,000,000 x 150)	2 R	✓	✓	
DFCM [30]		✓	Cloud base			-----	(1,024 x 10)	1 R	✓	✓	
PFMGPU [36]	✓		GPU-MPI			Intel® Core™ i7 3.4 GHz, 16 GB RAM, GPU NVidia GeForce GTS 450 con 4 SMs	(40,646 x 2)	2 R 1 S		✓	✓
brFCM [31]	✓		GPU	✓		Intel® Core™ i5 Compilador GCC 4.7, Ubuntu OS, Tesla M2070 y Tesla K20m	262,144 *píxeles*	2 R	✓		
PBCFCM1 PBCFCM2 [32]	✓		GPU		✓	Intel® Core™ i5-3230M, 2.6 GHz, NVidia GeForce GT 740m, 2 GB GPU	7,929,856 *píxeles*	4 R			✓
DP/HCG/ HNP [33]	✓		GPU		✓	Intel® Core™ i7, 8 GB RAM, NVidia GeForce GT 840M, Windows 8	-----	2 R	✓		✓
PFCM [34]	✓		GPU			Intel® Core™ i7 32 GB RAM, NVidia GTX 970 6 GB	-----	1 R	✓		
K-Means FM/FCM [37]	✓		GPU			Intel Xeon® Silver 4216, 3.20 GHz, Nvidia RTX 2080 Ti	(100,000 x 80)	1 S	✓	✓	
FM FCM GK-FCM [38]	✓		GPU			Intel® Skylake-X™ i7-37820X, 3.60 GHz. Nvidia A100, 2.4 GHz. Nvidia V100, 2.7 GHz	(800,000 x 104)	1 S	✓	✓	
PIT2FCM_1 [39]	✓		GPU		✓	Intel® Core™ i5-3230M, 2 cores, 2.6 GHz, NVidia GeForce GT 740 m 2 GB	1,331,200 *píxeles*	9 R			✓
PFCM [40]	✓		Spark			Intel® Xeon™ E7-4820 2.00 GHz, 8 GB RAM, 600 GB disco duro	2.5 GB	6 R			
FCM-Ck [41]	✓		Spark	✓	✓	Intel® Core™ i5-8400, 16 GB RAM, 2 TB disco duro, Windows 7	(400,000 x 42)	3 R	✓	✓	
ICFCM [42]	✓		Map Reduce	✓	✓	CPU 2.40 GHz, 2 GB RAM, Ubuntu 14.10	-----		✓	✓	
HCFCM [43]	✓		Map Reduce	✓	✓	CPU 2.2 GHz, 4 GB RAM, 500 GB disco duro, Ubuntu 12.10	(1,728 x 4)	2 R	✓		
FCM [44]	✓		Map Reduce	✓		AMD, 8 GB RAM, 500 GB disco duro, Ubuntu 18.04 OS	1 GB	6 S	✓		
FPGA [45]	✓		OpenCL			Intel® Core™ i7-6700 3.4 GHz, 16 Gb RAM	(100,000 x 2)	1 R 3 S	✓		
FCM [26]	✓		*OpenMP*			Intel Core 2	241,200 *píxeles*	1 R	✓		
PHFCM [27]	✓		*OpenMP*			Intel® Core™ i7-4510U con 4 cores, 2.00 GHz, 8 GB RAM	2,073,600 *píxeles*	4 R		✓	
POFCM	✓		***OpenMP***	✓	✓	Intel® Core™ i5-4570 con 4 cores, 3.30 GHz, 14 GB RAM y AMD® RyzenTM 5 5500 con 12 cores a 3.60 GHz, 112 GB RAM	**(40,000 x 40)**	**3 R 16 S**	✓		✓

a) De igual manera, en la mayoría de las investigaciones que se presentan, solo se resuelven datasets reales.

Cabe señalar que no se realizó ninguna comparación de tiempo o alguna otra métrica debido a que muchos artículos no presentan resultados claros o los presentan en función de distintas métricas. Además, una de las métricas más comunes la cual es la aceleración, no se puede comparar debido a que cada implementación realizada en los artículos tiene distintos procesadores o distintas GPU.

Capítulo 3

SELECCIÓN DE LA VARIANTE

En este capítulo, se presentan los motivos para realizar la selección de la variante de FCM a paralelizar. En la Sección 3.1, se menciona que se realizó una comparación entre algunas variantes de FCM. En la Sección 3.2, se presenta cómo está estructurada la variante seleccionada. Por último, en la Sección 3.3, se muestra la estructura del algoritmo HOFCM el cual es la variante seleccionada.

3.1 Comparación de variantes de FCM

Como primer paso, se implementó el algoritmo FCM, identificando cada una de sus fases principales, con el objetivo de seleccionar una variante de FCM enfocada en reducir el tiempo de solución de grandes *datasets*. A continuación, se muestran las cinco principales fases del algoritmo FCM:

1. Inicialización: se seleccionan al azar los valores iniciales de la matriz de pertenencia.
2. Cálculo de centroides: se calcula la ubicación de los centroides con base en la pertenencia de cada objeto con respecto a los centroides.
3. Clasificación: se asigna un valor de pertenencia de cada objeto con respecto a cada centroide, dependiendo la distancia al objeto.
4. Criterio de paro: para cada objeto se calcula la máxima diferencia de la matriz de pertenencia actualizada con la de la iteración anterior, este valor

se compara con un umbral, si es menor se detiene el algoritmo, de otra manera se repiten los pasos 2,3 y 4.

Existen mejoras en las distintas fases del algoritmo FCM [19], Una de las fases que posee mayor número de mejoras es la de inicialización, debido a que FCM es un algoritmo que es sensible a esta fase, debido a que en esta fase se seleccionan los centroides iniciales.

Se exploraron diversas mejoras de FCM y se encontraron relevantes las siguientes variantes: NFCM, FCM ++, *Improved* Canopy-FCM y HOFCM las cuales mejoran la fase de inicialización proporcionando resultados relevantes en la mejora de calidad del agrupamiento y en la reducción del tiempo de ejecución del algoritmo.

3.2 Variante seleccionada

Debido a que HOFCM presenta en la mayoría de los casos, mejores resultados en reducción de tiempos y calidad de solución, que el resto de algoritmos. De acuerdo con la tabla que se muestra en [19]. Se decidió seleccionar como algoritmo a paralelizar HOFCM considerando, adicionalmente, los excelentes resultados en tiempo y calidad que en la mayoría de los casos presenta, al resolver grandes instancias de datos, reales y sintéticas. Como ya se mencionó anteriormente, en HOFCM se mejora la selección inicial de grados de la matriz de grados de pertenencia al realizar unos pasos extras en la fase de inicialización.

HOFCM utiliza tres algoritmos para su ejecución, K++, OKM y FCM, a continuación, se presenta una descripción junto con un pseudocódigo de estos algoritmos.

3.2.1 Algoritmo K++

El algoritmo K++ que se propone en [24] busca optimizar la fase de inicialización para K-Means al seleccionar objetos del *dataset* a ejecutar, como centroides. En donde los objetos más lejanos entre sí, tienen mayor probabilidad de ser elegidos

como nuevos centroides. A continuación, en el Algoritmo 1, se muestra el pseudocódigo de K++.

Algoritmo 1: K++

Inicialización:
  X: = $\{x_1, \ldots, x_n\}$;
  Asignar el valor para k;
  V: = {Ø};
Selección:
  Seleccionar aleatoriamente de manera uniforme el centroide k_1 V: =X_i;
  for i = 2 to k:
    Seleccionar el i centroide v_i de X con probabilidad:

$$D(x_i, v_j)/\sum_{x \epsilon X} D(x_i, v_j);$$

    V: = X_i;
  Fin de *for*
  Devolver V
Fin del algoritmo

Para el Algoritmo 1, la inicialización comienza al leer el *dataset* X el cual se va a ejecutar, posteriormente, se asigna el valor de k que es el número total de centroides. En la fase de selección el primer paso es seleccionar de forma aleatoria un objeto del conjunto X y asignarlo como primer centroide k_1. Para la selección del segundo centroide hasta k se seleccionará i centroide con una probabilidad ponderada según la distancia entre los objetos, teniendo mayor probabilidad de ser seleccionado como siguiente centroide el objeto más lejano a los centroides previamente seleccionados.

3.2.2 Algoritmo OKM

El siguiente algoritmo que se utiliza es OKM, el cual es propuesto en [25]. Este algoritmo tiene el propósito de lograr que el algoritmo K-Means realice menos iteraciones. Esto se logra al parar el algoritmo cuando la cantidad de objetos que cambia su pertenencia de un conjunto a otro es menor a un umbral previamente establecido. Este umbral, busca el balance entre la relación de esfuerzo computacional y calidad de solución. A continuación, en el Algoritmo 2 se muestra el pseudocódigo de OKM.

Algoritmo 2: OKM

```
1   Inicialización:
2       X: = {x1, ..., xn};
3   Asignar el valor para k;
4       V: = {v1, ..., vk};
5       εok: = Umbral para determinar la convergencia de OKM;
6   Clasificación:
7       For xi ∈ X and vk ∈ V {
8           Calcular la distancia euclidiana de cada xi a cada vk;
9           Asignar el xi objeto a el centroide vk más cercano;
10          Computar γ};
11  Calcular centroides:
12      Calcular el centroide vk;
13  Convergencia:
14      Si ( γ ≤ εok):
15          Parar el algoritmo;
16      De otra manera:
17          Ir a la Clasificación
18  Fin del algoritmo
```

Para el Algoritmo 2, la inicialización comienza al leer el *dataset* X el cual se va a ejecutar. Después, se asigna el valor de *k* que es el número total de centroides y se seleccionan aleatoriamente los centroides de v_1 hasta v_k. Por último, se asigna el valor εok el cual es el umbral de convergencia para OKM.

En la fase de clasificación, para cada objeto se calculará la distancia euclidiana a cada centroide y se asignará al centroide más cercano. Como último paso de esta fase se calculará γ el cual representa la cantidad de objetos que cambió de grupo en una iteración *t*, el cual es calculado como $\gamma_t = 100(o_t/n)$, donde o_t es el número de objetos que cambiaron de grupo. En la fase de cálculo se calcularán los centroides de v_1 hasta v_k.

Por último, se evaluará si se cumple el criterio de paro, si se cumple se detiene el algoritmo, de otra manera se repiten todos los pasos desde la fase de clasificación.

3.2.3 Algoritmo FCM

Para FCM, la agrupación de datos, se logra mediante un proceso de optimización iterativo que minimiza una función objetivo. La función objetivo que es minimizada se muestra a continuación:

$$J_m(U,V) = \sum_{i=1}^{n}\sum_{j=1}^{c}(u_{ij})^m \left|\left|x_i - v_j\right|\right|^2 \quad (1)$$

Para esta función $X=\{x_1, \ldots, x_n\}$ es el conjunto de n datos a particionar, donde $x_i \in R^d$ para i=1, …, n; c es el número total de grupos, donde $2 \leq c < n$; $V=\{v_1, \ldots, v_c\}$ es el grupo de centroides, donde v_j es el centroide del grupo j; m es el exponente de ponderación o factor difuso que indica cuanto se traslapan los grupos, $m > 1$; $\left|\left|x_i - v_j\right|\right|^2$ es la distancia entre el objeto x_i y el centroide v_j; por último, $U=u_{ij}$ es el grado de pertenencia de cada objeto i a cada grupo j; sujeta a:

$$\sum_{j=1}^{c} u_{ij} = 1 \quad (2)$$

Fuzzy C-Means logra la optimización de J_m a través de los cálculos iterativos de u_{ij} y v_j, utilizando las siguientes ecuaciones:

$$v_j = \frac{\sum_{i=1}^{n}(u_{ij})^m x_i}{\sum_{i=1}^{n}(u_{ij})^m} \quad (3)$$

y

$$u_{ij} = \left(\sum_{k=1}^{c}\left(\frac{d_{ij}}{d_{ik}}\right)^{\frac{2}{m-1}}\right)^{-1} \quad (4)$$

Donde x_i y v_j son vectores que pertenecen al espacio R^d y se representan como:

$$x_i = (x_1, x_2, \ldots, x_d), \quad 1 \leq i \leq n \tag{5}$$

$$v_j = (v_1, v_2, \ldots, v_d), \quad 1 \leq j \leq c \tag{6}$$

En [15], se presenta el algoritmo FCM en Fortran, esta implementación sirve como base para trabajos posteriores en FCM y es similar al algoritmo que se conoce hoy en día. A continuación, en el Algoritmo 3 se observa el pseudocódigo de FCM.

Algoritmo 3: FCM estándar

```
1   Inicialización:
2       X: = {x1, …, xn};
3       Asignar el valor para c;
4       ε: = Umbral para determinar la convergencia;
5       U(t): = {μ11, …, μij}; se genera aleatoriamente
6   Calcular centroides:
7       Calcular los centroides con Eq. (3);
8   Clasificación:
9       Actualizar y calcular la matriz de membresía con Ec. (4);
10  Convergencia:
11      Si max [abs(μij(t) − μij(t+1))] < ε:
12          Parar el algoritmo;
13      De otra manera:
14          U(t): = U(t+1) y t: = t + 1;
15      Ir al cálculo de centroides
16  Fin del algoritmo
```

Para el Algoritmo 3, en la fase de inicialización (línea 2) se lee el *dataset* X, el cual se va a resolver; en la línea 3 se asigna el valor de *c* que es el número total de centroides; en la línea 4 se asigna el valor del criterio de paro (épsilon) el cual señala qué tan precisa se espera que sea la solución; en la línea 5 se genera de manera aleatoria la matriz de membresía donde $U^t=\{u_{11}, \ldots, u_{ij}\}$; en la línea 6 se encuentra la fase de cálculo de centroides, la cual en la línea 7 calcula los centroides con la Ecuación 3; en la línea 8 se encuentra la fase de clasificación

para la cual se calcula y actualiza la matriz de pertenencia en la línea 9 mediante la Ecuación 4; en la línea 10 se encuentra la fase de criterio de paro, para la línea 11 se evalúa si la máxima diferencia absoluta de cada valor de la matriz de pertenencia de la iteración actual con respecto a la anterior es menor que el criterio de paro (épsilon), por lo cual, si se cumple esta condición, el algoritmo se detiene en la línea 12, de otra manera se actualiza la matriz de pertenencia con los nuevos valores en la línea 14 y se vuelven a realizar todos los pasos desde la fase de cálculo de centroides hasta que se cumpla el criterio de paro y se pare el algoritmo.

3.3 HOFCM

Con la implementación de los tres anteriores algoritmos, se logró comprender el funcionamiento de la variante HOFCM el cual es un algoritmo híbrido que optimiza la fase de inicialización al seleccionar mejores centroides. Con lo que logra reducir el tiempo de ejecución de FCM y mejora la calidad de los resultados. A continuación, en la Figura 2, se presenta un diagrama del algoritmo HOFCM.

En el diagrama de la Figura 2, se observa el flujo de HOFCM, primero se tiene la fase de inicialización en donde se carga la base de datos a ejecutar y se asignan los valores iniciales, posteriormente se inicia un bucle que va desde i=1 hasta i<=t donde t=10, en el artículo [19] se menciona que se selecciona este valor para t con base en resultados experimentales, dentro del bucle se ejecuta el algoritmo K++, los centroides obtenidos de K++ se utilizan en la fase de inicialización de OKM, al concluir el bucle se compara el resultado de la función objetivo de cada ejecución de OKM en la fase de optimizar centroides, el conjunto de los centroides de la ejecución con la mejor función objetivo se transformará en matriz de pertenencia en la fase de función s y por último los grados de pertenencia resultantes se ingresaran a FCM para generar el agrupamiento difuso.

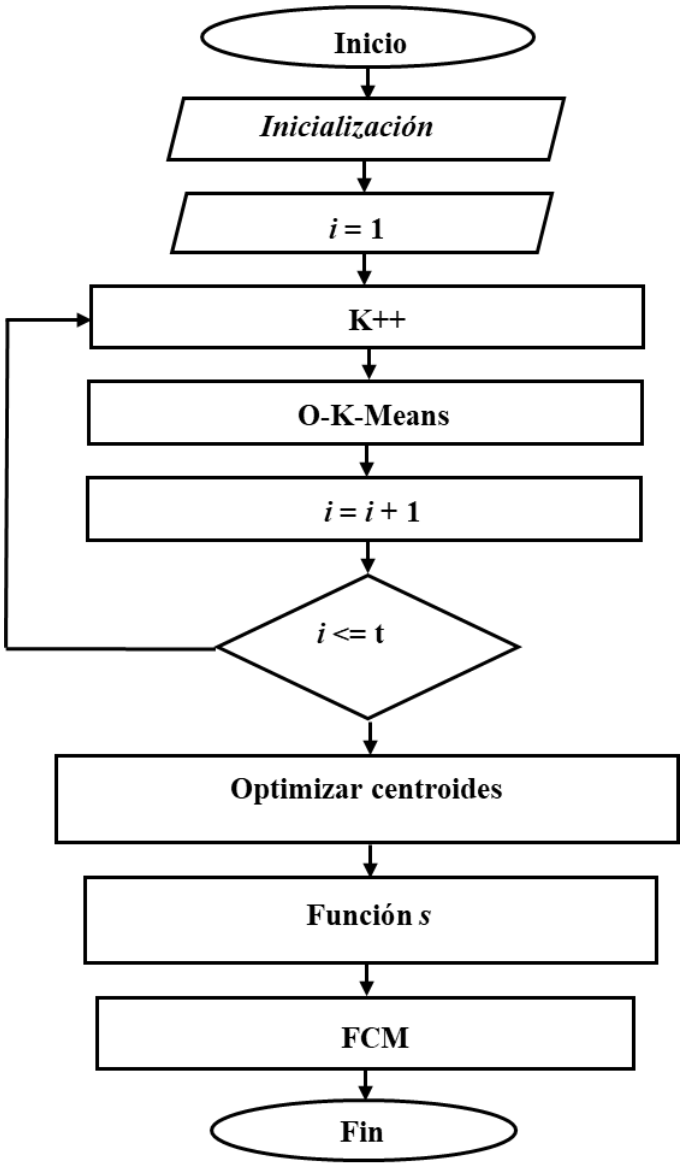

Figura 2. Diagrama del algoritmo HOFCM.

Con estos pasos se logra ejecutar HOFCM, para comprobar los resultados de esta variante del algoritmo FCM se implementó HOFCM en las mismas condiciones que se describen en la Sección 5.1.1, y se realizaron algunas pruebas, las cuales se muestran a continuación en la Tabla 2.

Tabla 2. Pruebas con HOFCM y FCM estándar para el dataset Spam

SPAM			FCM estándar		HOFCM	
Grupos (c)	Objetos (n)	Dimensión (d)	Tiempo promedio (s)	Función objetivo (millones)	Tiempo promedio (s)	Función objetivo (millones)
2	4601	57	1.70	632.46	1.40	632.47
4	4601	57	8.14	228.92	5.63	222.94
6	4601	57	73.42	141.73	9.51	110.32
8	4601	57	127.85	101.05	23.05	70.58
10	4601	57	110.23	80.16	26.89	41.95
14	4601	57	375.06	42.37	97.11	20.69
18	4601	57	765.12	35.32	167.30	13.44
26	4601	57	2,220.28	23.72	447.49	7.16

Como se observa en estas pruebas la comparativa con FCM estándar muestra que el algoritmo obtiene resultados relevantes en la reducción de tiempo para la mayoría de los casos, especialmente cuando se ejecuta con un valor para *c* mayor.

Capítulo 4

PARALELIZACIÓN DE LA VARIANTE SELECCIONADA

En esta Sección se muestra cómo se paralelizó la variante de FCM seleccionada en la Sección anterior. En la Sección 4.1, se presenta un pequeño resumen del método para paralelizar dicha variante. Este método es *OpenMP*. En la Sección 4.2, se presentará el diagrama de flujo y el pseudocódigo de la variante paralela POFCM.

4.1 Paralelización con *OpenMP*

Para implementar con un enfoque paralelo la variante seleccionada se utilizó la librería *OpenMP*, la cual es un método de paralelización que utiliza memoria compartida. Se decidió utilizar este método, debido a los excelentes resultados que ha presentado al paralelizar, en los cuales de acuerdo a [46] ha mostrado ser superior a otros métodos para paralelizar. A continuación, se detallan algunas de las principales ventajas al paralelizar con *OpenMP*:

1. Es un método para paralelizar portátil y reutilizable, debido a que se puede utilizar en diversas plataformas y sistemas operativos sin necesidad de realizar considerables modificaciones al código.

2. *OpenMP* utiliza directivas que se pueden implementar en un código existente sin necesidad de realizar grandes modificaciones, por lo que es simple de implementar.
3. La paralelización con *OpenMP* es escalable, debido a que se pueden utilizar distintos equipos con distinta cantidad de hilos y como se mencionó en el punto 1, se puede ejecutar en un equipo con mayor cantidad de núcleos sin necesidad de modificar el código.
4. Este método de paralelización utiliza memoria compartida para la comunicación entre hilos lo que brinda un código más limpio, sin necesidad de directivas complejas, esto implica menos errores.

Por otra parte, algunas de las desventajas de *OpenMP* son:

1. Debido a que *OpenMP* utiliza memoria compartida la comunicación entre hilos puede generar problemas de coherencia en caché. Por esta razón, aunque sea simple de implementar se debe tener cuidado al generar los *pragmas*.
2. *OpenMP* brinda poco control cuando se paralelizan las tareas en los hilos, por lo que no es recomendable implementarlo en aplicaciones que requieran un mayor control sobre el comportamiento de los hilos
3. Esta API de paralelización solo está disponible para ciertos lenguajes como lo que limita su uso para otros lenguajes.

Aunque *OpenMP* es un método de paralelización muy útil y proporciona excelentes resultados con poco esfuerzo, se deben tener en cuenta sus ventajas y desventajas para comprender en qué situaciones es especialmente útil. Considerando sobre todo la paralelización con memoria compartida como una ventaja, si se busca utilizar únicamente un equipo de cómputo y obtener excelentes resultados.

OpenMP permite generar aplicaciones paralelas utilizando programación paralela con memoria compartida. Los lenguajes de programación en los cuales se puede implementar *OpenMP* son C, C++ y Fortran. Esta técnica permite paralelizar un conjunto de instrucciones contenido dentro de un bucle mediante

directivas denominadas *pragmas* las cuales dividen el total de veces que se repetirá la instrucción, dentro del bucle *for*, en los hilos con los que cuente el equipo (*fork*). Posteriormente, al terminar el bucle, todos los hilos devuelven la información al núcleo maestro (*join*). A continuación, en la Figura 3, se muestra una representación gráfica de cómo funciona esta paralelización por bucles.

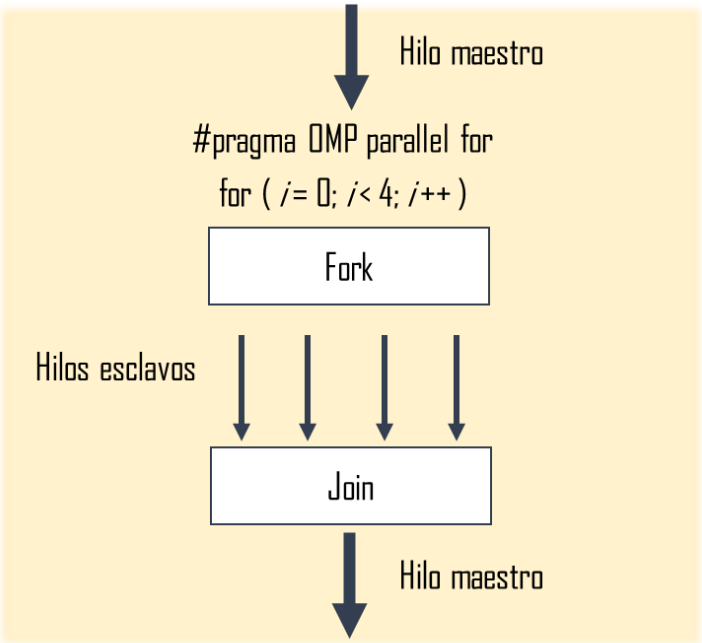

Figura 3. Representación de los hilos para *OpenMP*.

Como se observa en la Figura 3, la división de las tareas entre los hilos reduce considerablemente el tiempo total de procesamiento. En este sentido, se ha observado que *OpenMP* es escalable debido a que se pueden paralelizar los bucles de forma transparente dividiendo una tarea entre el número total de hilos que posea el equipo.

Como se mencionó anteriormente las directivas utilizadas para paralelizar los bucles son llamadas *pragmas*, éstos brindan una instrucción al hilo maestro para dividir la tarea en el bucle *for* para los hilos esclavo. Para explicar de forma más simple estas directivas en el Programa 1, se muestra un ejemplo sobre un código paralelizado con *OpenMP* en lenguaje C, este código se extrajo de [46].

Programa 1: Ejemplo *#pragma*

```
1   void ejemplo (double * a, double* b) {
2   int n=11, i;
3   #pragma omp parallel for
4       for (i = 1; i < n+1; i ++) {
5           b[i] = (a[i] + a[i-1]) / 2;}
6   }
```

Como se observa en el Programa 1, en la línea 1, se inicia el *void* "*ejemplo"*; en la línea 2, se define y asigna la variable *n*=11; en la línea 3, se observa la directiva *pragma* la cual paraleliza la ejecución de las siguientes instrucciones mediante el bucle *for*; en la línea 4, la directiva pragma asigna dentro del bucle un bloque o conjunto de iteraciones a cada hilo. De esta manera, si la máquina tiene cuatro hilos, el primero procesará las iteraciones 1,2 y 3; el segundo hilo procesará las iteraciones 4,5 y 6; el tercer hilo procesará las iteraciones 7, 8 y 9; y finalmente, el cuarto hilo procesará las iteraciones 10 y 11. Por este procedimiento, será reducido de manera considerable el tiempo de ejecución al utilizar cuatro procesadores para ejecutar lo que antes se ejecutaba con uno.

Existen diversas cláusulas en *OpenMP*, las principales que fueron consideradas para la presente investigación se enlistan a continuación:

a) Num_threads: Indica la cantidad de hilos que se utilizaran en la región paralela.
b) Shared: Indica que una variable será compartida por todos los hilos en la región paralela.
c) Private: Indica que una variable será privada para cada hilo en la región paralela.
d) Schedule: Indica cómo se asignarán las iteraciones de un bucle a los hilos.

e) Reduction: Indica que una operación será aplicada a una variable en todos los hilos y que el resultado será almacenado en una variable privada para cada hilo, al final de la región paralela, los resultados parciales de todas las variables privadas se unifican para obtener un resultado final.

Estas cláusulas brindan la posibilidad de crear regiones paralelas que ejecuten de forma correcta programas más complejos, en donde existe más de un bucle y variables que deben ser tratadas de forma específica. A continuación, en el Programa 2 se muestra un ejemplo de un código paralelo en lenguaje C que utiliza estas cláusulas.

Programa 2: Ejemplo 2 *#pragma con cláusulas*

```
1   void ejemplo2 ( ) {
2   num_threads (4);
3   int n=11, m=8, i, j;
4   long double a=0, b=0, c=0;
5   #pragma omp parallel for shared(i) private(j,a,b) reduction(+:c)
6       for (i = 1; i < n+1; i ++) {
7           for (j = 1; j < m+1; j ++) {
8           b = i + b;
9           a = j + a;
10           }
11       c += a + b;
12      }
```

En el Programa 2, se observa lo siguiente: En la línea 1, se inicia el *void* "*ejemplo 2"*; en la línea 2, se utiliza la cláusula *num_threads* para establecer la cantidad de hilos para la región paralela, en este caso se asignan cuatro hilos; en la línea 3, se definen y asignan los valores a las variables *i*, *j*, *n*=11 y *m*=8; en la

línea 4, se definen y asignan los valores a las variables *a*=0, *b*=0 y *c*=0; en la línea 5, se observa la directiva *pragma* la cual paraleliza la ejecución de las siguientes instrucciones mediante el bucle *for*. También se observan algunas cláusulas donde *shared* indica que la variable *i* del bucle principal se compartirá entre los distintos hilos, por lo que el primer hilo procesará las iteraciones 1, 2 y 3; el segundo hilo procesará las iteraciones 4, 5 y 6; el tercer hilo procesará las iteraciones 7, 8 y 9; y finalmente, el cuarto hilo procesará las iteraciones 10 y 11. La cláusula *private* indica que las variables *j*, *a* y *b* tendrán valores privados para cada hilo, donde cada hilo utilizará los valores de *j* del 1 al 8. Cada hilo, además, tendrá su propia variable *a* y *b* con valores privados. Por último, la cláusula *reduction* indica que para la variable *c,* se realizará una sumatoria en donde cada hilo almacenara de forma individual los valores de *c* en la región paralela y cuando esta finalice, se unirán los valores privados de cada hilo para *c* y se generara un resultado único.

4.2 Algoritmo propuesto: POFCM

La mejora propuesta, utiliza un enfoque paralelo e híbrido con base en el algoritmo HOFCM. En la Figura 4, se muestra un diagrama del algoritmo POFCM en el cual se identifican tres regiones paralelas que comprenden los tres algoritmos que se utilizaron para implementar esta variante.

La primera región contiene la paralelización de K++. La segunda comprende la paralelización de OKM. Ambas regiones paralelas están dentro de un bucle que se repite *i* veces hasta *t*=10. Para POFCM, se seleccionó el mismo valor utilizado para HOFCM, debido a que se realizaron experimentos con diversos valores de *t* y se observó que *t*=10 proporcionó buenos resultados. De estas *i* ejecuciones de regiones paralelas se busca obtener los resultados de la función objetivo y centroides de OKM. Al finalizar el bucle en la Sección de “selección de centroides” se seleccionarán los centroides de OKM de la ejecución en la cual se obtuvo el mejor valor para la función objetivo.

Por último, en la tercera región en la cual se paraleliza la función *s*, que transforma los centroides de OKM en una matriz de pertenencia, con la que posteriormente, se ejecutará FCM. Con esto se concluye la ejecución de POFCM.

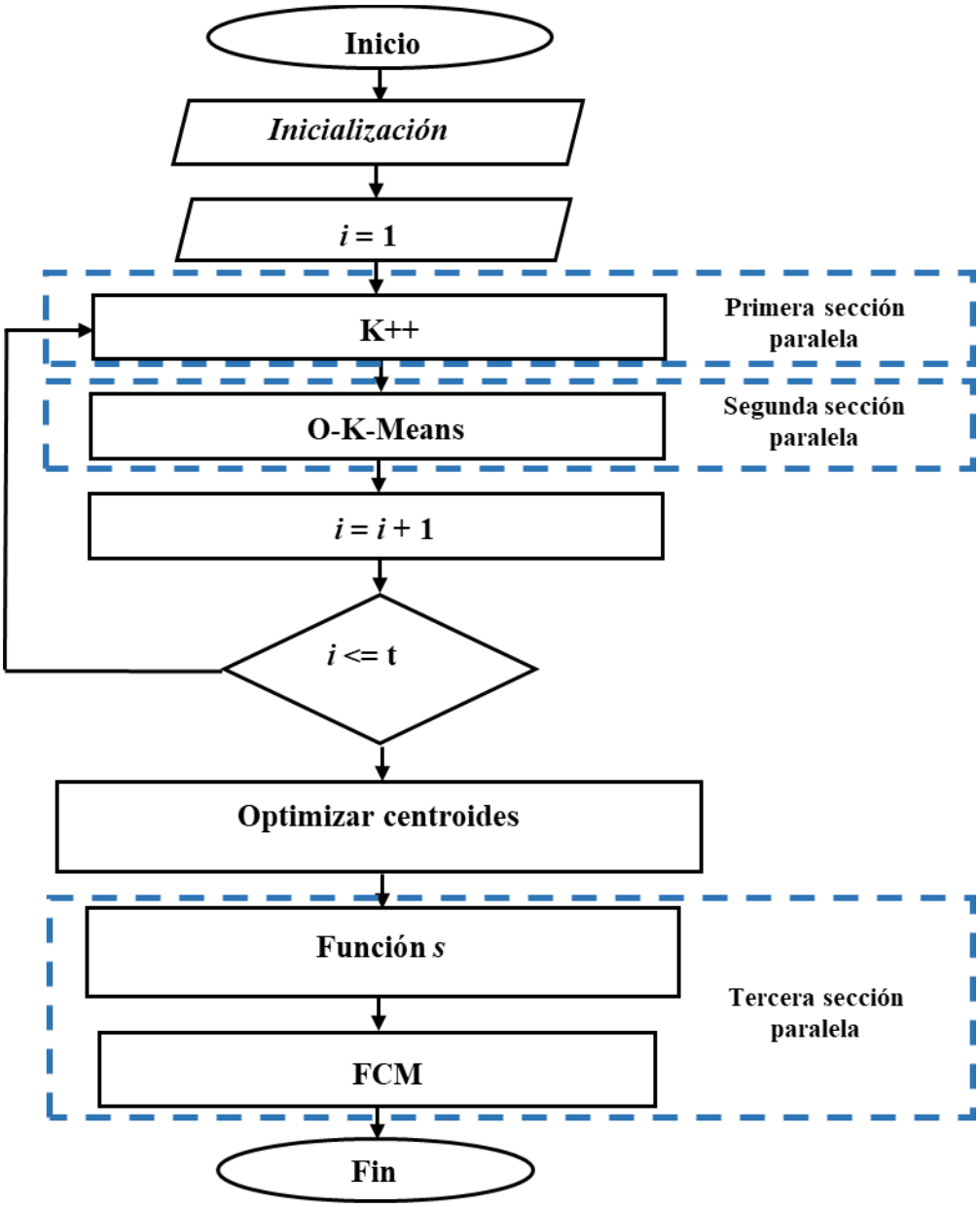

Figura 4. Diagrama del algoritmo POFCM

En la Figura 4, se presentó de manera simple el algoritmo POFCM mediante un diagrama. A continuación, en el Algoritmo 5, se muestran de manera detallada las regiones paralelas, dentro del algoritmo POFCM, el cual se separó

en cuatro fases. Las fases se dividen en inicialización, cálculo de centroides, clasificación y convergencia. Para POFCM los parámetros de entrada son: el conjunto de datos X; El número de grupos *c*; El exponente de ponderación *m*; el valor de umbral para FCM ε (criterio de paro para FCM); El valor de umbral para OKM εok (criterio de paro para OKM) y el número de hilos. Los parámetros de salida son: el conjunto de centroides V y los valores de la matriz de pertenencia U.

Como ya se mencionó, la codificación paralela para POFCM se realizó con *OpenMP*. Las regiones paralelas que se muestran en la Figura 4, de manera general y con mayor detalle en el Algoritmo 5, incluyen las directivas pragma justo antes del primer bucle *for* que se utiliza en cada una de las funciones anidadas que contienen los algoritmos K++, OKM y la función s. Además, se incluyen cláusulas específicas para garantizar el procesamiento adecuado dentro del entorno paralelo. Por último, en las funciones que conforman el algoritmo de FCM las cuales son "Centroides", "Clasificación" y "Convergencia" se incluyen también las directivas pragma antes del primer bucle *for* junto con las cláusulas necesarias para su funcionamiento.

Algoritmo 5: POFCM

Entrada: *dataset X, c, m, ε, εok, T(num_threads);*
Salida: *V, U;*

```
1   Inicialización:
2       i: = 1;
3       Repetir
4           Función K++ (X, c, T):  // Esta función es paralela.
5             Devolver V';
6           Función OKM (X, V', εok, c, T):  // Esta función es paralela.
7             Devolver V'',FO;
8           i = i + 1;
9       Mientras i <= t;
10      Seleccionar V'' para el  i en el cual el valor de la función objetivo FO sea el
11      Función s (X, V'', m, T);  // Esta función es paralela.
12          Devolver U;
13      r: = 1;
14  Calcular centroides:
15  Función Centroides (X, U, m, T);  // Esta función es paralela.
16      //Calcular los centroides, con Eq. (2);
```

17 Devolver V;
18 **Clasificación:**
19 Función Clasificación (X, V, m, T); **// Esta función es paralela.**
20 //Actualizar y calcular la matriz de membresia, con Eq. (3);
21 Devolver U;
22 **Convergencia:**
23 Función de Convergencia (X, U, ε, T, r, *paro*); **// Esta función es paralela.**
24 **Si** max [abs($\mu_{ij}^{(t)} - \mu_{ij}^{(t+1)}$)] < ε **entonces**:
25 *paro* := 0; //se active la bandera de paro del algoritmo.
26 Devolver U, *paro*
27 **De otra manera**:
28 $U^{(r)}$: = $U^{(r+1)}$ y r: = r + 1;
29 *paro* := 1;
30 Devolver U, *paro*
31 **Si** *paro* = 1 **entonces:** ir a **calcular centroides**
32 **fin del algoritmo**

Como se muestra en el Algoritmo 5, se tienen cuatro fases, las cuales se detallan a continuación:

La primera fase es la de *inicialización*, la cual abarca desde la línea 1 hasta la línea 13. En la línea 3, se genera un bucle que se repite *i* veces hasta *t*=10, este bucle termina en la línea 9. Dentro del bucle se encuentran dos funciones: la función K++ paralela en la línea 4, la cual devuelve el conjunto de centroides V' y la función OKM paralela, la cual se muestra en la línea 6, esta función devuelve el conjunto de centroides V'' y la función objetivo FO. Al finalizar el bucle en la línea 10, se realiza la selección de los centroides de OKM de la iteración *i* en la cual se obtuvo el mejor valor para la función objetivo. El último paso de la fase de inicialización es el transformar los centroides a valores de la matriz de pertenencia U mediante la función s en la línea 11, la cual devuelve la matriz U.

La segunda fase es *calcular centroides*, esta fase comprende desde la línea 14 a la 17, en la línea 15, se muestra la función de centroides la cual está paralelizada, este cálculo de centroides se realiza con la Ecuación 2, descrita en el apartado 3.2.3, esta fase devuelve el conjunto de centroides V.

La tercera fase es *clasificación*, se muestra desde la línea 18 a la 21. En la línea 19, se encuentra la función clasificación, la cual es paralela. Esta función

calcula los valores de la matriz de pertenencia utilizando la Ecuación 3, la cual se encuentra en el apartado 3.2.3, esta fase devuelve los valores de la matriz de pertenencia U.

La cuarta y última fase es la de *convergencia*, que abarca desde la línea 22 a la 32. En la línea 23, se muestra la función de convergencia paralelizada. Esta fase, realiza los cálculos para obtener el valor absoluto máximo de los valores de la matriz de membresía en dos iteraciones consecutivas, como se observa en la línea 24. Si el valor obtenido es mejor que el umbral ε se asigna el valor de 0 a la variable de paro, se devuelven los valores de la matriz U y el valor de paro. De lo contrario, se actualizan los valores de la matriz de membresía U, se incrementa el contador de iteraciones r, se asigna el valor de 1 a la variable de paro y se devuelven los valores de la matriz U y la variable de paro. Por último, como se observa en la línea 31, si la variable de paro tiene el valor de 1 el algoritmo vuelve a la fase de “calcular centroides”.

Capítulo 5

DISEÑO DE LOS EXPERIMENTOS

En este capítulo se describen los experimentos que se realizaron con POFCM con los que se muestra que se logró una mejora paralela significativa. En la Sección 5.1, se describe el entorno paralelo para realizar los experimentos. En la Sección 5.2, 5.3 y 5.4 se describen el primer, el segundo y el tercer experimento respectivamente, junto con los *datasets* que se utilizaron en cada experimento. Por último, en la Sección 5.5, se muestran las métricas que se tomaron en cuenta para evaluar el desempeño de la mejora paralela.

5.1 Entorno de los experimentos

Para evaluar el rendimiento de la mejora propuesta, se diseñaron tres experimentos. El primero, permitió evaluar el rendimiento de la mejora paralela, propuesta, para el cual se utilizaron instancias de datos reales.

En el segundo experimento, se utilizaron bases de datos sintéticas, las cuales se crecieron en objetos y dimensiones de manera controlada. El objetivo de este experimento fue observar tendencias y constatar los resultados de la mejora en un entorno con mayor control.

Para el tercer experimento, se utilizaron instancias de gran tamaño como las que se presentan en el *bigdata,* el fin de este experimento es mostrar la escalabilidad del algoritmo al implementarse en un equipo de cómputo más robusto y observar el comportamiento del algoritmo con instancias de datos de gran tamaño.

Para los dos primeros experimentos, se utilizó el mismo equipo de cómputo. En el caso del tercer experimento, se utilizó un equipo más robusto.

5.1.1 Entorno del primer y segundo experimento

En el primer experimento, se implementaron los siguientes algoritmos secuenciales: HOFCM y FCM estándar y el algoritmo paralelo POFCM. Para el segundo experimento, solo se implementó el algoritmo secuencial HOFCM y el algoritmo paralelo POFCM. La paralelización de POFCM se realizó con *OpenMP* 2.0. Todos los algoritmos fueron implementados en lenguaje C con el compilador de Microsoft Visual C++ (MSVC) 2020. El equipo de cómputo que se utilizó para los dos primeros experimentos tiene un procesador Intel® Core™ i5-4570 a 3.30 GHz, 14 GB de RAM, 512 GB almacenamiento HDD con sistema operativo Windows 10.

Las bases de datos reales utilizadas para las pruebas del primer experimento fueron: Abalone, Spam y Urban, obtenidas del repositorio de la Universidad de California, UCI Machine Learning Repository. En el caso del segundo experimento, se generaron bases de datos sintéticas y se crecieron de manera controlada en dimensiones, objetos y centroides a ejecutar con el fin de observar el comportamiento del algoritmo.

5.1.2 Entorno del tercer experimento

Para observar la escalabilidad de POFCM y su comportamiento con bases de datos de gran tamaño, se diseñó el tercer experimento, para el cual se ejecutaron HOFCM secuencial y POFCM paralelo. Para paralelizar POFCM se utilizó *OpenMP* 2.0, los algoritmos se implementaron en lenguaje C con el compilador de Microsoft Visual C++ (MSVC) 2020. El equipo de cómputo utilizado es una PC con un procesador AMD® Ryzen™ 5 5500 a 3.60 GHz, 112 GB de RAM, 512 GB almacenamiento SSD con sistema operativo Windows 10.

Se generaron y utilizaron bases de datos sintéticas de gran tamaño y se crecieron de manera controlada en cantidad de objetos, con el objetivo de

observar el comportamiento del algoritmo al incrementar la cantidad de objetos y el peso del *dataset*.

Para la ejecución de los algoritmos, se utilizaron los siguientes parámetros: factor difuso m = 2, criterio de paro épsilon ε = 0.01 y para los algoritmos HOFCM y POFCM el umbral de paro para OKM εok= 0.72. Los valores iniciales de la membresía de pertenencia fueron seleccionados de manera aleatoria para el caso de FCM estándar. En el caso de HOFCM y POFCM se utilizaron los mismos valores para la matriz de pertenencia, los cuales fueron generados al transformar los centroides obtenidos de OKM en grados de pertenencia como se muestra en la Sección 3.3.

5.2 Descripción del primer experimento

Para la ejecución de ambos algoritmos se utilizaron los siguientes parámetros: factor difuso m = 2, criterio de paro épsilon ε = 0.01 y el umbral de paro para OKM εok= 0.72. Para ambos algoritmos se utilizaron los mismos valores para la matriz de pertenencia, los cuales fueron generados al transformar los centroides obtenidos de OKM en grados de pertenencia como se muestra en la Sección 3.3.

5.2.1 Descripción de las instancias de datos

Las instancias de datos reales que se utilizaron para realizar las pruebas del primer experimento se enlistan a continuación en la Tabla 3 en donde se muestra el número de datos n, su dimensión d y el tamaño de $n*d$.

Tabla 3. Instancias reales para las pruebas del primer experimento

Nombre	*Objetos*	*Dimension (d)*	Indicador $n*d$
Abalone	4,177	7	29,239
Spam	4,601	57	262,257
Urban	360,177	2	720,354

Para las instancias reales se utilizaron 2, 4, 6, 8 ,10, 14, 18 y 26 grupos, con cada tamaño de los grupos se realizaron 10 pruebas. Los resultados que se presentan son el promedio de esas 10 ejecuciones.

5.3 Descripción del segundo experimento

En este experimento se estableció una comparación ente HOFCM y POFCM utilizando métricas que evalúan la paralelización. Estas métricas se detallan en la Sección 5.5. Estos experimentos se realizan con instancias de datos sintéticas que crecen de forma controlada con el fin de observar tendencias al incrementar la cantidad de datos, de dimensiones y de grupos para las ejecuciones del algoritmo.

Con el objetivo de demostrar la ventaja de tiempo que presenta la mejora propuesta al resolver instancias reales. Adicionalmente, se comparará POFCM con HOFCM para obtener métricas que evalúan el rendimiento paralelo.

5.3.1 Descripción de las instancias de datos

Las instancias de datos sintéticas que se utilizaron para realizar las pruebas del segundo experimento se enlistan a continuación en la Tabla 4, en la primera Columna, se muestra el ID del *dataset*, en la segunda Columna, se encuentra el nombre, en la tercera Columna, los objetos del *dataset n*, en la cuarta Columna, su dimensión *d* y por último en la quinta Columna, el tamaño de *n***d*.

Tabla 4. Instancias reales para las pruebas del primer experimento.

ID	Nombre	*n*	*d*	Indicador *n***d*
1	*S10_10*	10,000	10	100,000
2	*S10_20*	10,000	20	200,000
3	*S10_30*	10,000	30	300,000
4	*S10_40*	10,000	40	400,000
5	*S20_10*	20,000	10	200,000
6	*S20_20*	20,000	20	400,000
7	*S20_30*	20,000	30	600,000
8	*S20_40*	20,000	40	800,000
9	*S30_10*	30,000	10	300,000
10	*S30_20*	30,000	20	600,000
11	*S30_30*	30,000	30	900,000
12	*S30_40*	30,000	40	1,200,000
13	*S40_10*	40,000	10	400,000
14	*S40_20*	40,000	20	800,000
15	*S40_30*	40,000	30	1,200,000

16	*S40_40*	40,000	40	1,600,000

Como se observa en la Tabla 4, se crearon 16 bases de datos. Para estas instancias sintéticas se utilizaron 2, 12, 22 y 32 grupos.

5.4 Descripción del tercer experimento

Para el último experimento, se realiza la comparación entre HOFCM y POFCM con bases de datos sintéticas de gran tamaño. Estas bases de datos se generaron de tal modo que se incrementa de manera controlada la cantidad de objetos que contienen, sin modificar la cantidad de características que tienen los objetos.

El objetivo de este experimento, fue mostrar que la mejora propuesta puede ser utilizada en instancias de datos de gran tamaño, con lo cual, mantiene una clara ventaja en las métricas que evalúan la paralelización, las cuales se detallan en la siguiente sección.

5.4.1 Descripción de las instancias de datos

Las instancias de datos sintéticas que se utilizaron para realizar las pruebas del tercer experimento, se enlistan a continuación en la Tabla 5, en la primera Columna, se muestra el ID del *dataset*; en la segunda Columna, se encuentra el nombre del *dataset*; en la tercera Columna, los objetos del *dataset n*; en la cuarta Columna, su dimensión *d*; en la quinta Columna, el tamaño de *n***d* y, por último, en la sexta Columna, se muestra su tamaño en disco.

Tabla 5. Instancias reales para las pruebas del primer experimento.

ID	Nombre	*n*	*d*	Indicador *n***d*	Tamaño
13	*S5m*	5,000,000	40	200,000,000	6.5 GB
14	*S10m*	10,000,000	40	400,000,000	13 GB
15	*S13m*	13,000,000	40	520,000,000	16.9 GB
16	*S26m*	26,000,000	40	1,040,000,000	33.9 GB
17	*S78m*	78,000,000	40	3,120,000,000	101 GB
18	*S156m*	156,000,000	40	6,240,000,000	203 GB

Como se observa en la tabla anterior, se crearon seis bases de datos de gran tamaño. Para estas instancias sintéticas se utilizaron dos grupos. Se destaca el tamaño de las últimas tres instancias, las cuales superan los 30 GB de peso en disco y los mil millones para el indicador n*d.

5.5 Métricas de evaluación

Para evaluar el algoritmo paralelo se utilizaron dos métricas:

La primera métrica, calcula la aceleración o *speedup* (Sp) y se utiliza para evaluar la velocidad del algoritmo paralelo, porque indica cuántas veces más rápido resuelve el problema en comparación con el algoritmo secuencial. Esta métrica, está relacionada con la cantidad de hilos que tiene el equipo y un valor óptimo para esta métrica es $Sp = h$ donde h equivale al número de hilos del equipo. El cálculo de esta métrica, se realiza al dividir el tiempo secuencial (Ts) entre el tiempo paralelo (Tp), como se muestra en la Ecuación 7.

La segunda métrica, calcula la eficiencia paralela (Ep) y se utiliza para evaluar la calidad de la paralelización, porque indica qué tan paralelo es el algoritmo, siendo 1, el valor óptimo [47]. El cálculo de esta métrica, se obtiene al dividir el *speedup* (Sp) entre el número de hilos (h) como se muestra en la Ecuación 8.

$$Sp = \frac{Ts}{Tp} \tag{7}$$

$$Ep = \frac{Sp}{h} \tag{8}$$

Capítulo 6

RESULTADOS EXPERIMENTALES DE POFCM

En este capítulo, se presentan los resultados de los tres experimentos realizados, así como su análisis. En la Sección 6.1, 6.2 y 6.3, se muestran los resultados de los experimentos 1, 2 y 3, respectivamente. Por último, en la Sección 6.4 se presenta el análisis de los resultados de los experimentos mencionados anteriormente, y las observaciones relevantes.

6.1 Resultados del primer experimento

Como ya se ha mencionado, el primer experimento consistió en ejecutar tres *datasets* reales con los algoritmos: FCM estándar, HOFCM y POFCM. Cada uno de estos *datasets* se ejecuta para ocho distintas cantidades de grupos. Con cada cantidad de grupos se realizaron 10 ejecuciones y el promedio de esas ejecuciones es lo que se reporta.

A continuación, en la Tabla 6, 7 y 8, se observan los resultados de este experimento para Abalone, Spam y Urban, respectivamente. Para todas las tablas, se tiene la siguiente disposición: en la primera Columna, se muestra la cantidad de grupos con los que se ejecutó el *dataset*; en la segunda Columna, se muestra el tiempo de ejecución promedio de FCM estándar para cada cantidad de grupos; en la tercera Columna, se muestra el tiempo promedio que tardo HOFCM para cada cantidad de grupos; en la cuarta Columna, se muestran los resultados de

HOFCM, en donde se muestra el porcentaje de reducción de tiempo para cada cantidad de grupos; en la quinta Columna, se muestra el tiempo promedio que tardo POFCM para cada cantidad de grupos; la sexta Columna, muestra el porcentaje de reducción de tiempo de POFCM para cada cantidad de grupos y por último, la séptima Columna, contiene el porcentaje de ganancia en calidad para POFCM el cual es el mismo para HOFCM y POFCM.

Tabla 6. Resultados del primer experimento para Abalone

ABALONE	FCM estándar	**HOFCM**		**POFCM**		
Grupos (c)	Tiempo promedio (s)	Tiempo promedio (s)	% reducción de tiempo	Tiempo promedio (s)	% reducción de tiempo	% ganancia de calidad
2	0.11	0.32	-179.01	0.17	-48.78	0.01
4	0.82	1.05	-27.30	0.50	39.46	0.08
6	3.34	2.01	39.76	0.77	76.95	0.14
8	5.80	4.21	27.32	1.50	74.20	0.98
10	8.69	9.37	-7.89	2.99	65.55	-0.02
14	26.79	15.92	40.57	4.69	82.49	-0.01
18	37.31	29.62	20.60	8.50	77.22	-0.02
26	138.78	70.19	49.43	19.49	85.95	-0.05

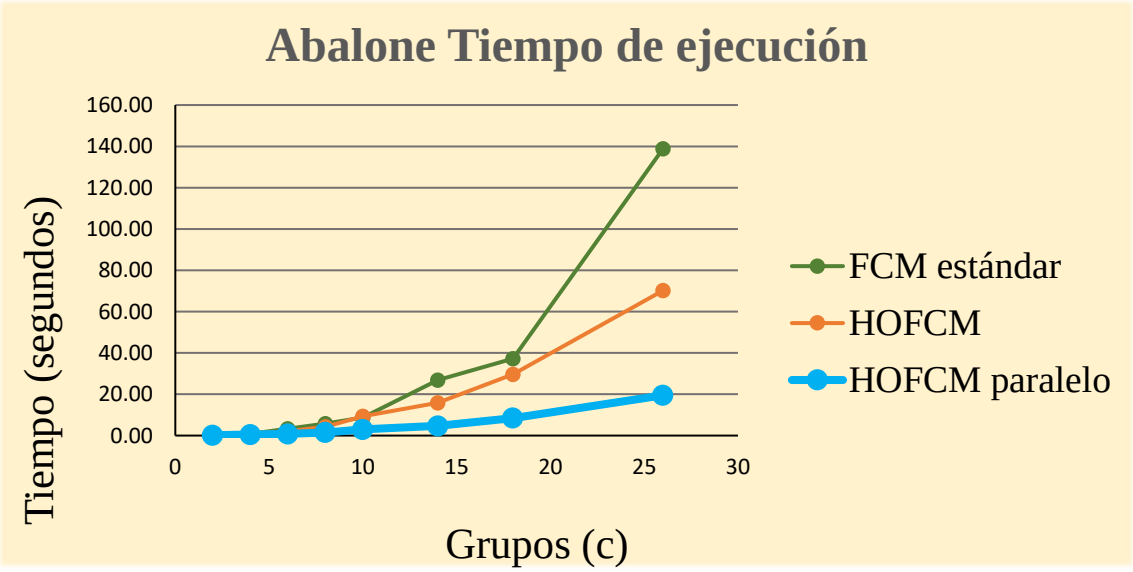

Figura 5. Tiempo de ejecución para el primer experimento para Abalone

En la Figura 5, se muestra una gráfica que muestra el tiempo de ejecución de FCM estándar, HOFCM y POFCM. Como se observa, con una mayor cantidad de grupos la ejecución de POFCM consume significativamente menos tiempo que la de HOFCM y FCM estándar.

Tabla 7. Resultados del primer experimento para Spam

SPAM	FCM estándar	**HOFCM**		**HOFCM paralelo**		
Grupos (c)	Tiempo promedio (s)	Tiempo promedio (s)	% reducción de tiempo	Tiempo promedio (s)	% reducción de tiempo	% ganancia de calidad
2	1.70	1.40	17.45	0.79	53.69	0.00
4	8.14	5.63	30.85	2.43	70.10	2.61
6	73.42	9.51	87.05	3.69	94.98	22.16
8	127.85	23.05	81.97	7.12	94.43	30.15
10	110.23	26.89	75.61	8.55	92.24	47.67
14	375.06	97.11	74.11	27.71	92.61	51.17
18	765.12	167.30	78.13	46.47	93.93	61.94
26	2,220.28	447.49	79.85	123.03	94.46	69.81

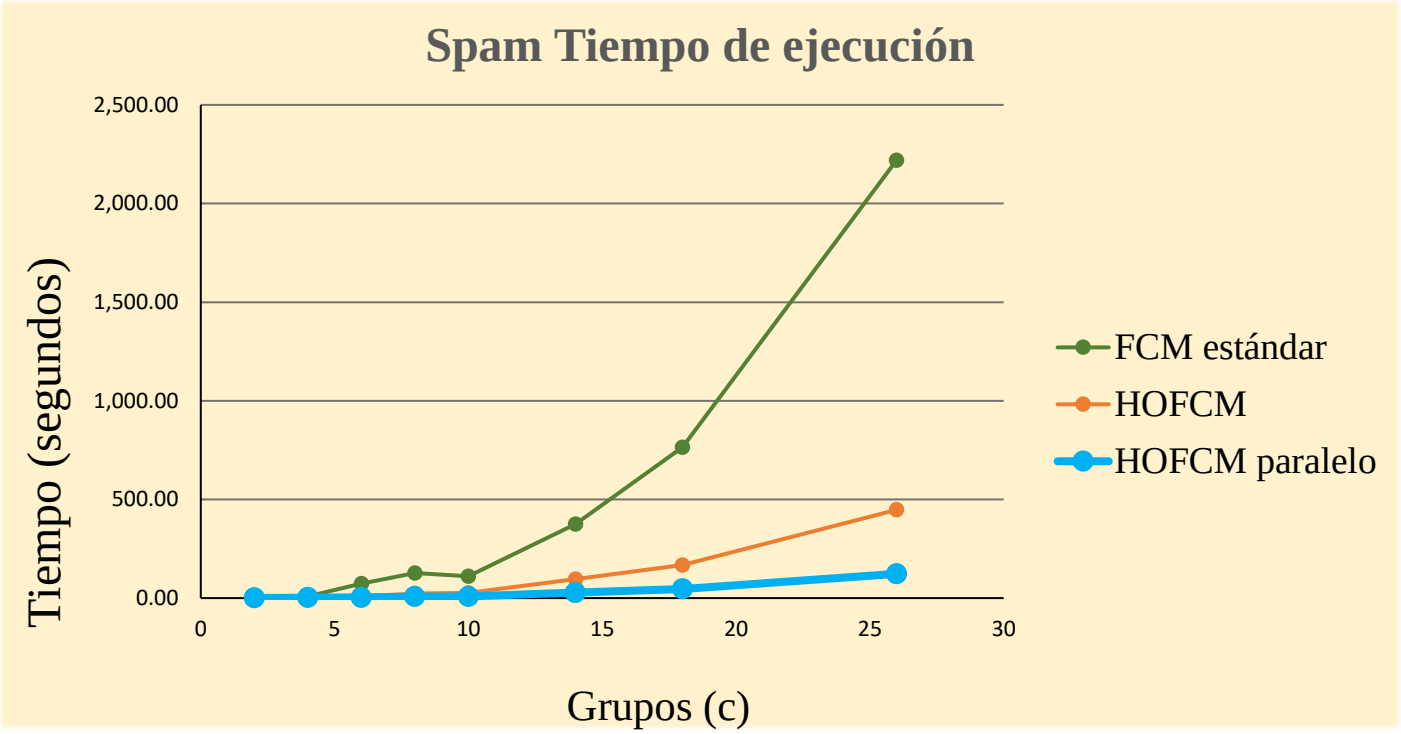

Figura 6. Tiempo de ejecución para el primer experimento para Spam

En la Figura 6, se muestra la gráfica comparativa del tiempo de ejecución de FCM estándar, HOFCM y POFCM para el *dataset* Spam. Con una mayor cantidad de grupos, la ejecución de POFCM consume significativamente menos tiempo que ejecutar HOFCM y FCM estándar.

Tabla 8. Resultados del primer experimento para Urban

URBAN	FCM estándar	**HOFCM**		**HOFCM paralelo**		
Grupos (c)	Tiempo promedio (s)	Tiempo promedio (s)	% reducción de tiempo	Tiempo promedio (s)	% reducción de tiempo	% ganancia de calidad
2	3.34	14.19	-325.46	7.61	-128.02	0.00
4	36.22	29.46	18.65	11.90	67.15	0.00
6	75.24	62.37	17.10	22.34	70.31	2.65
8	109.59	166.67	-52.08	49.70	54.65	0.78
10	221.14	129.97	41.23	41.70	81.15	4.98
14	442.83	256.28	42.13	74.34	83.21	7.17
18	750.12	542.62	27.66	152.86	79.62	8.50
26	1,685.87	1,143.19	32.19	304.09	81.96	12.84

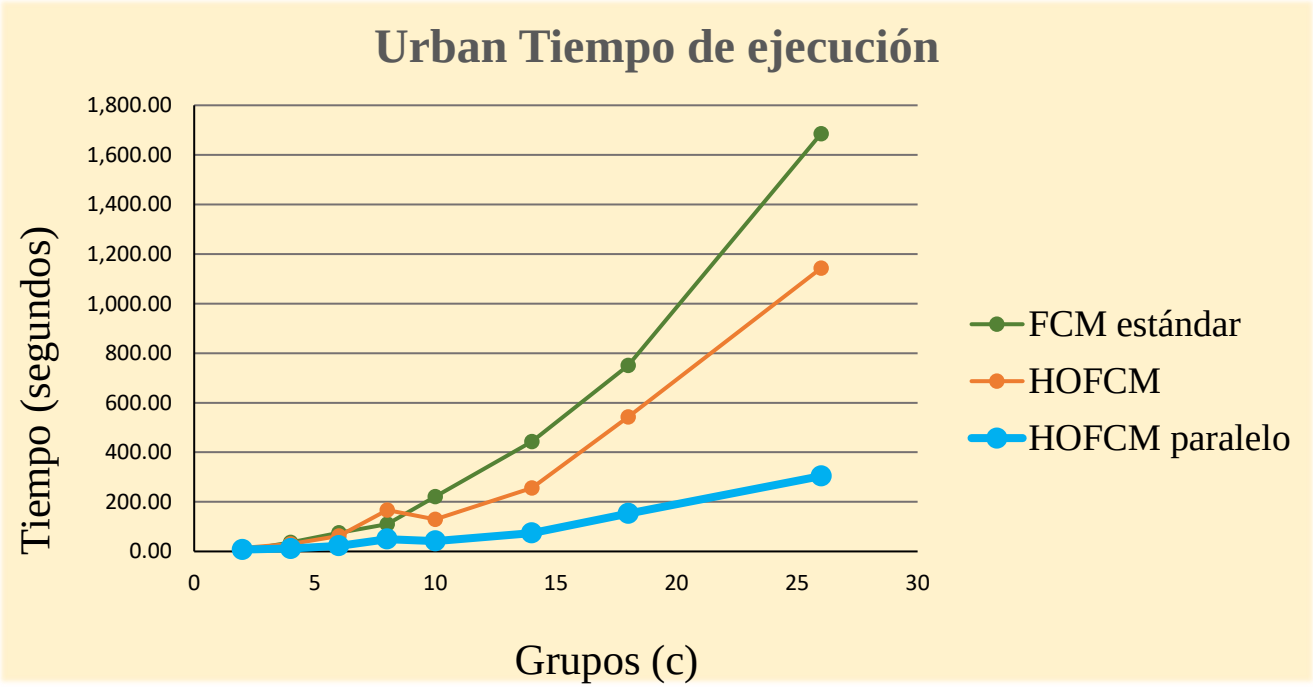

Figura 7. Tiempo de ejecución para el primer experimento para Urban

En la Figura 7, la gráfica muestra el tiempo de ejecución de FCM estándar, HOFCM y POFCM para el *dataset* Urban. Como se observa, se mantiene la tendencia de que la reducción de tiempo de POFCM sea más significativa cuando hay mayor cantidad de grupos.

Para este experimento, también se comparó HOFCM con POFCM para medir su desempeño paralelo por medio de las métricas que se presentan en la Sección 5.5. A continuación, en la Tabla 9, se presentan las métricas de aceleración Ecuación 8 y eficiencia paralela Ecuación 9.

Tabla 9. Primer experimento, métricas de evaluación paralela entre HOFCM y POFCM

Grupos (c)	ABALONE		SPAM		URBAN	
	Aceleración (*speedup*) Sp	Eficiencia Paralela Ep	Aceleración (*speedup*) Sp	Eficiencia Paralela Ep	Aceleración (*speedup*) Sp	Eficiencia Paralela Ep
2	1.88	0.47	1.78	0.45	1.87	0.47
4	2.1	0.53	2.31	0.58	2.48	0.62
6	2.61	0.65	2.58	0.64	2.79	0.7
8	2.82	0.7	3.24	0.81	3.35	0.84
10	3.13	0.78	3.14	0.79	3.12	0.78
14	3.39	0.85	3.5	0.88	3.45	0.86
18	3.49	0.87	3.6	0.9	3.55	0.89
26	3.6	0.9	3.64	0.91	3.76	0.94

Como se muestra en la Tabla 9, conforme aumenta la cantidad de grupos los resultados de aceleración y eficiencia paralela se acercan al óptimo, logrando valores mayores a 0.9 en eficiencia paralela.

Por último, para mostrar de manera visual los resultados de aceleración para los tres *datasets* se muestra la Figura 8, en la cual se aprecia una tendencia de que conforme se ejecuta el *dataset* con más grupos se acerca la eficiencia paralela al valor óptimo.

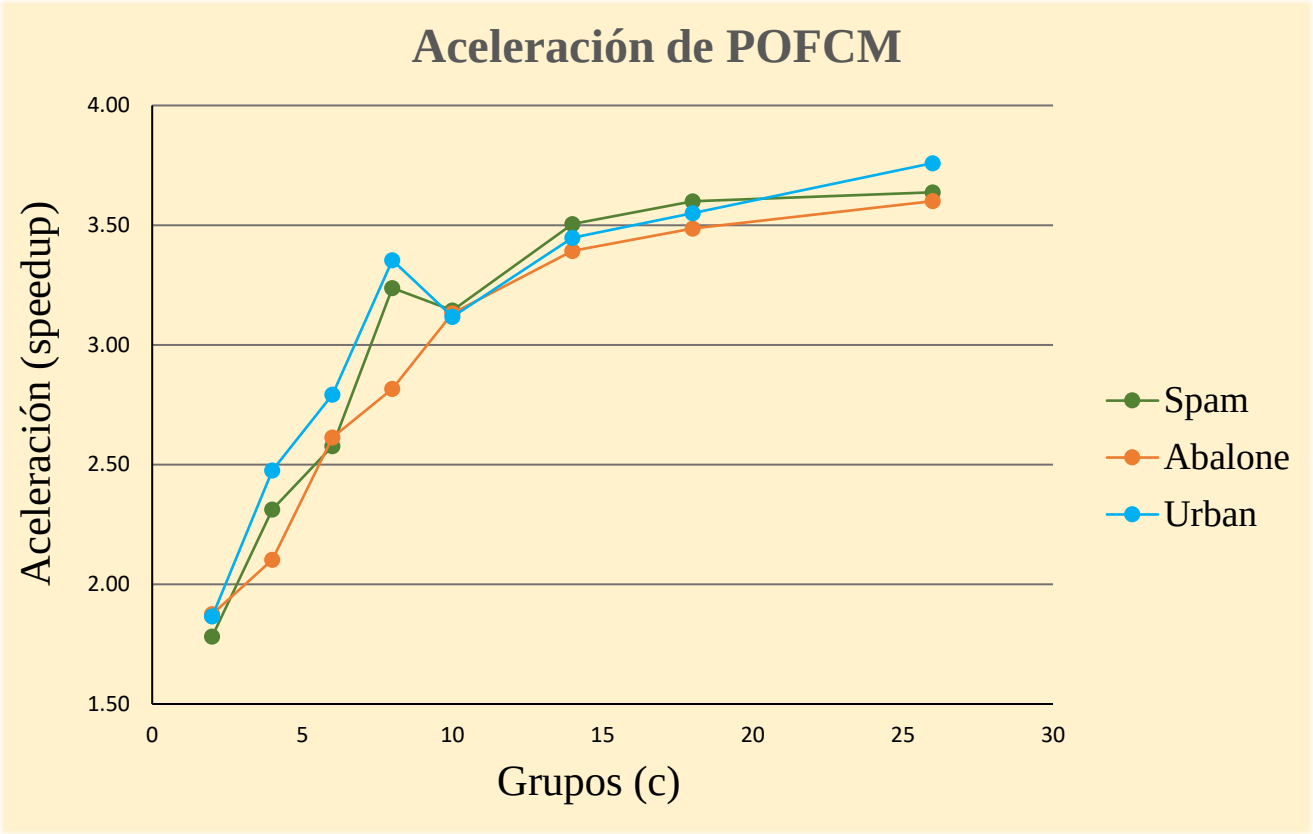

Figura 8. Aceleración de POFCM para los *datasets* del experimento 1

6.2 Resultados del segundo experimento

Para el segundo experimento, se ejecutaron 16 *datasets* sintéticos con los algoritmos HOFCM y POFCM cada *dataset*, se ejecuta para cuatro distintas cantidades de grupos.

En la Tabla 10, se muestra una recopilación de las ejecuciones para dos de los *datasets* más pequeños y dos de los *datasets* más grandes. La distribución de esta tabla es la siguiente: en la primer Columna, se muestra el nombre del *dataset*; en la segunda Columna, la cantidad de grupos; en la tercer Columna, los objetos; en la cuarta Columna, las dimensiones; en la quinta Columna, se muestra el tiempo de ejecución de HOFCM; en la séptima Columna, se observa el tiempo de ejecución de POFCM; por último, en la octava Columna, se presenta el valor de la eficiencia paralela el cual se calcula con la Ecuación 9.

Tabla 10. Comparativa de POFCM y HOFCM para 4 *datasets* sintéticos del segundo experimento.

Nombre	Grupos (c)	Objetos (n)	Dimensión (d)	HOFCM Tiempo(s)	POFCM Tiempo(s)	Eficiencia paralela
S10_10	2	10,000	10	2.85	1.71	0.42
S10_10	12	10,000	10	22.50	8.09	0.70
S10_10	22	10,000	10	57.32	17.15	0.84
S10_10	32	10,000	10	86.83	25.07	0.87
S10_40	2	40,000	10	9.60	5.30	0.45
S10_40	12	40,000	10	107.00	36.15	0.74
S10_40	22	40,000	10	283.36	84.86	0.83
S10_40	32	40,000	10	407.80	116.89	0.87
S40_10	2	10,000	40	17.85	7.27	0.61
S40_10	12	10,000	40	102.58	29.23	0.88
S40_10	22	10,000	40	188.06	49.37	0.95
S40_10	32	10,000	40	295.37	75.96	0.97
S40_40	2	40,000	40	74.93	29.08	0.64
S40_40	12	40,000	40	600.57	155.88	0.96
S40_40	22	40,000	40	1,055.79	268.34	0.98
S40_40	32	40,000	40	1,584.59	400.44	0.99

Como se aprecia en la Tabla 10, la eficiencia paralela de POFCM aumenta conforme crece la complejidad del *dataset* a ejecutar. Estos *datasets* sintéticos se crearon precisamente con el fin de observar estas tendencias. En la Fila 4, con 22 grupos se obtiene una eficiencia paralela superior a 0.8. Se observa que para los dos *datasets* con 10 dimensiones, la aceleración superior a 0.8 se presenta cuando la cantidad de grupos es superior a 22. Por otra parte, en los *datasets* con 40 dimensiones, la aceleración es superior a 0.8 cuando hay 12 o más grupos. Otro punto relevante es que como se muestra en la fila 12 aún con 10,000 objetos, 40 dimensiones y 22 grupos, se logra una aceleración superior a 0.9, a diferencia de en la Fila 8, en la que con 40,000 objetos, 10 dimensiones y 10 grupos se obtiene una aceleración inferior a 0.9.

Por lo que se concluye que, el factor que más influye para que la eficiencia paralela aumente, es la cantidad de grupos con los que se ejecuta el *dataset*. el segundo factor más importante, es la cantidad de dimensiones que tiene el *dataset*. Por último, el valor menos relevante para aumentar la eficiencia paralela es la cantidad de objetos.

Adicionalmente, para mostrar de manera gráfica la tendencia que se mencionó, se presenta en la Figura 9, una gráfica con la aceleración de POFCM cuando se ejecutan los 16 *datasets*, cada uno con las cuatro distintas cantidades de grupos. Como se mencionó en la Sección 5.5, el valor óptimo para la aceleración es igual al número de hilos que tiene el equipo el cual en este caso es cuatro.

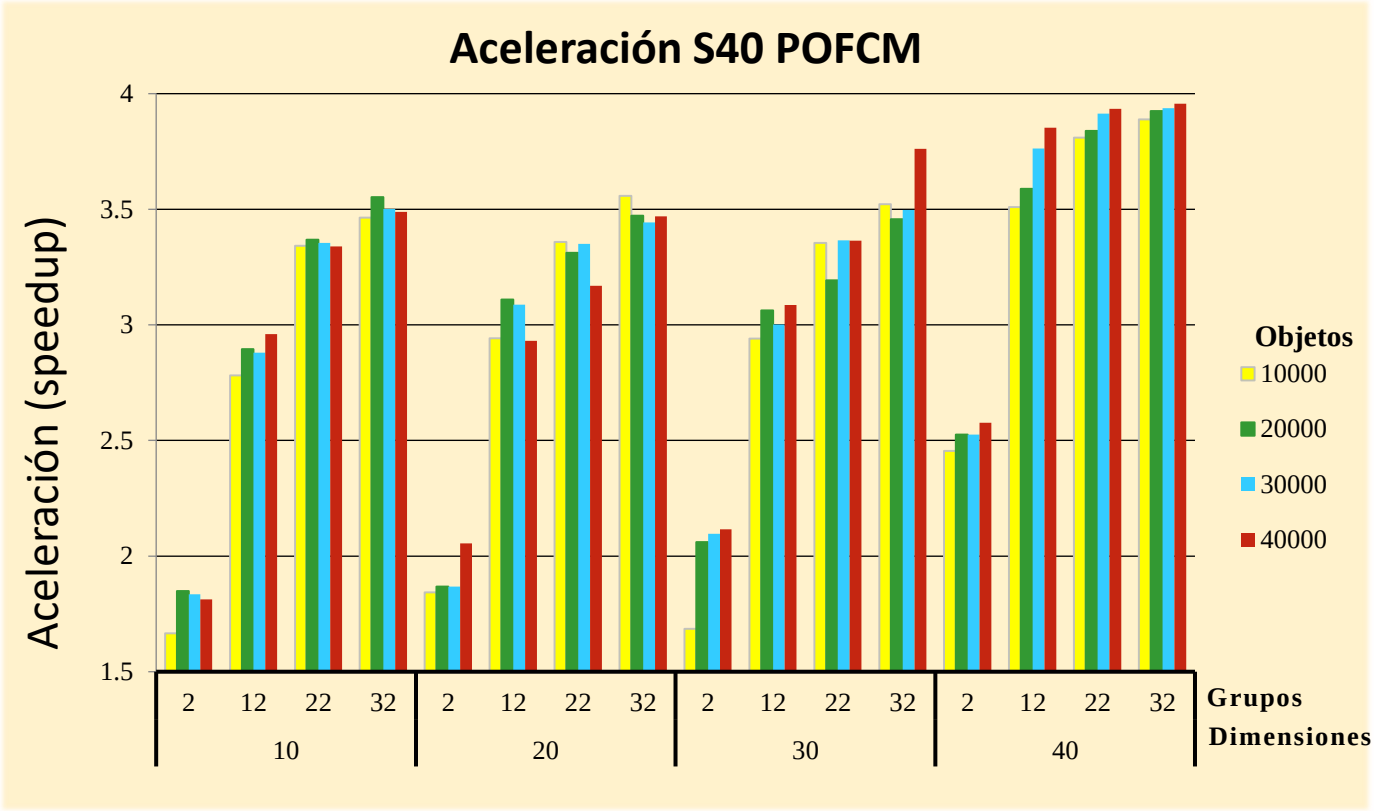

Figura 9. Aceleración de POFCM para los 16 *datasets* sintéticos del segundo experimento.

6.3 Resultados del tercer experimento

El tercer experimento, consiste en mostrar que el algoritmo propuesto POFCM presenta mejores resultados al ser ejecutado con un *dataset* de gran tamaño como los del *bigdata*.

En la Tabla 11, se muestran los experimentos realizados, el contenido tiene la siguiente distribución: en la Columna 1, se muestra la cantidad de objetos del

dataset; en la Columna 2, el tamaño del *dataset*; en la Columna 3, el número de grupos; en la Columna 4, la cantidad de características de los datos; en la Columna 5, el tiempo de ejecución de HOFCM y en la Columna 6, el tiempo de ejecución de POFCM.

Tabla 11. Resultados del tercer experimento.

Objetos (n)	Tamaño	Grupos (c)	Dimensión (d)	% de Reducción de tiempo
5,000,000	7 GB	2	40	66.60
10,000,000	14 GB	2	40	65.51
13,000,000	19 GB	2	40	65.55
26,000,000	34 GB	2	40	65.43
78,000,000	101 GB	2	40	66.24
156,000,000	203 GB	2	40	65.48

Como se aprecia aún con los *datasets* de mayor tamaño, se obtuvo una reducción de tiempo significativa. Como ya se mencionó, este experimento se llevó a cabo con un equipo de cómputo más robusto, sin necesidad de modificar el código, como se muestra POFCM reduce el tiempo de ejecución cuando se procesan grandes instancias de datos.

6.4 Análisis y observaciones de los resultados de POFCM

En esta Sección, se analizan los resultados de los tres experimentos realizados con el algoritmo POFCM.

6.4.1 Análisis de los resultados del primer experimento

Con base en los resultados expuestos en la Sección 6.1 se destaca lo siguiente:

Para la comparativa de FCM estándar con POFCM:

1. Para los tres *datasets*, la reducción de tiempo se vuelve más significativa cuando se aumenta la cantidad de grupos.
2. Únicamente, cuando se ejecutó Abalone y Urban con dos grupos se obtuvo un tiempo de ejecución mayor, debido a que esta mejora requiere ejecutar

varias veces k++ y OKM y se vuelve poco eficiente cuando se tiene un *dataset* pequeño o pocos grupos.

3. En el mejor de los casos, POFCM redujo el tiempo de ejecución en un 94.98%

Para la comparativa entre HOFCM con POFCM:

1. Se determinó que las métricas de aceleración y eficiencia paralela tienden a aumentar conforme crece el número de grupos.
2. Es relevante destacar que la calidad de HOFCM y POFCM es la misma.
3. Conforme crece el número de grupos, los valores de eficiencia paralela y aceleración tienen a acercarse al valor óptimo.

6.4.2 Análisis de los resultados del segundo experimento

Considerando los resultados presentados en la Sección 6.2, se observa lo siguiente:

1. Al resolver los 16 conjuntos de datos, se observó que POFCM tiende a aumentar la aceleración y eficiencia paralela cuando se aumenta la cantidad de grupos y cuando se crece la dimensión del conjunto de datos.
2. En el mejor de los casos, para el conjunto de datos S40_40 con 32 grupos, se logró una aceleración de 3.96 y una eficiencia paralela de 0.99 el cual es un valor cercano al óptimo.

Es relevante mencionar, que en este experimento se diseñaron las bases de datos de tal manera que fueron creciendo de manera uniforme en cantidad de datos y dimensiones con el fin de observar tendencias.

6.4.3 Análisis de los resultados del tercer experimento

Considerando los resultados que se presentaron en la Sección 6.3, se aprecia lo siguiente:

1) Como se esperaba, el algoritmo POFCM reduce el tiempo de ejecución al ejecutarse con bases de datos de gran tamaño.

Es relevante destacar que en este experimento se crearon las bases de datos sintéticas de tal manera que el algoritmo POFCM realizara únicamente una ejecución, al crear los *datasets* con valores cercanos y únicamente seleccionar dos grupos.

6.4.4 Análisis comparativo

Como se muestra en la Sección 2, existen pocos trabajos sobre algoritmos basados en FCM paralelizados con *OpenMP*. De entre estos trabajos, solo se encontró uno con el que se pueden comparar los resultados presentados en las secciones anteriores. Para comparar los resultados obtenidos con POFCM se seleccionaron los mejores resultados reportados en [26] en donde los experimentos resuelven un conjunto de datos diez veces y reportan el promedio de esas ejecuciones. El tamaño del conjunto de datos fue de 241,200 *píxeles*, el número de grupos 16. Se señala que se obtuvo un promedio de 107.315 segundos con la versión secuencial y 55.092 segundos con la versión paralela, lo que equivale a una eficiencia paralela de 0.97 y una aceleración de 1.95 debido a que el equipo de pruebas tiene dos núcleos.

Por otra parte, los mejores resultados obtenidos con el segundo experimento, en donde se utilizó un conjunto de datos de 40,000 objetos con 40 dimensiones y 32 grupos son 1,584.6 segundos en la versión secuencial, 400.4 segundos en la versión paralela, lo que equivale a una eficiencia paralela de 0.99 y una aceleración de 3.96 debido a que el equipo de pruebas tiene cuatro núcleos.

Como se muestra, la implementación de POFCM presenta una mejor eficiencia paralela, al menos en los experimentos que se evaluaron. Además, es importante destacar que en el artículo [26] los experimentos se realizaron con un conjunto de datos mientras que con POFCM se han utilizado 25 conjuntos de datos.

Capítulo 7

CONCLUSIONES Y TRABAJOS FUTUROS

Como resultado de la presente investigación, se mostró que, mediante la implementación de programación paralela en una variante de FCM, es factible reducir el tiempo de procesamiento del algoritmo. La variante seleccionada es HOFCM la cual tiene enfoque en la solución de grandes *datasets*. Dicha variante fue rediseñada mediante el paradigma de programación paralela y se le denominó *Parallel Optimization Fuzzy C-Means* (POFCM). POFCM se implementó mediante la arquitectura de *OpenMP*. Los resultados de los experimentos muestran que la implementación obtuvo excelentes resultados en aceleración y eficiencia paralela, cuando se resolvieron *datasets* de gran tamaño.

En este capítulo, se exponen las conclusiones derivadas de la investigación, así como algunas recomendaciones para desarrollar trabajos futuros.

7.1 Aportaciones

La idea de rediseñar una variante de FCM surge de la necesidad de agrupar *dataset*s en menos tiempo del que requiere actualmente FCM, utilizando los mismos recursos computacionales. El enfoque de esta investigación fue utilizar de manera eficiente los hilos disponibles en un equipo de cómputo, para resolver una instancia de datos de gran tamaño en un tiempo razonable.

POFCM está implementado mediante la técnica de paralelización que brinda *OpenMP* permite implementar un algoritmo utilizando memoria compartida y brindando un código simple, portable y escalable.

Es destacable que para realizar los experimentos con POFCM se crearon 16 *datasets* sintéticos con los cuales se observaron tendencias. Además, se crearon seis *datasets* sintéticos de gran tamaño con el objetivo de observar el comportamiento del algoritmo al resolver *datasets* como las que se presentan en el *bigdata*.

7.2 Conclusiones

La implementación del algoritmo POFCM, mostró que se disminuye el tiempo de ejecución conservando la calidad de agrupamiento del algoritmo *HOFCM*.

Para validar el enfoque paralelo de esta investigación se diseñaron tres experimentos, se resolvieron *datasets* sintéticos y reales, algunos de ellos de gran tamaño. Con el objetivo de valorar el rendimiento se utilizaron métricas como la aceleración y la eficiencia paralela. A continuación, se destacan los resultados:

1. La implementación de POFCM en *OpenMP* es escalable y puede ser utilizado en distintos equipos de cómputo sin necesidad de modificar el código, como se mostró en el experimento 3.
2. La implementación propuesta utilizó todos los hilos disponibles en los equipos de cómputo utilizados para los experimentos, resolviendo tareas en paralelo con cada hilo.
3. Los mejores resultados en eficiencia paralela se obtuvieron con *datasets* con un indicador *ndc* grande, destacando que el valor más relevante es la cantidad de grupos (c), seguido de las dimensiones (d).
4. Con base en los resultados experimentales, se observó que la implementación de *POFCM* redujo el tiempo de solución con respecto a HOFCM en todos los casos. Utilizando el mismo equipo de cómputo.

5. El *speedup* en los *datasets* sintéticos y reales alcanzó valores alentadores, superiores a 3.6 utilizando cuatro hilos y una eficiencia paralela superior a 0.9.

6. Con relación a la calidad, comparando POFCM con FCM en el primer experimento, la calidad promedio para los tres *datasets* fue superior con POFCM presentando los mejores resultados para el *dataset* Spam con lo que se obtuvo una mejora en la calidad de en promedio un 35.68%.

7.3 Trabajos futuros

Para dar continuidad a este trabajo, se sugiere que, en investigaciones posteriores, se atiendan los siguientes temas:

a) Aplicar paralelismo distribuido a *POFCM* para utilizar una mayor cantidad de equipos computacionales al utilizar una red de computadoras.

b) Añadir una mejora en alguna otra fase del algoritmo POFCM, como podría ser la fase de convergencia.

REFERENCIAS

[1] A. S. Shirkhorshidi, S. Aghabozorgi, T. Y. Wah, and T. Herawan, 'Big data clustering: A review', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8583 LNCS, no. PART 5, pp. 707–720, 2014, doi: 10.1007/978-3-319-09156-3_49/COVER.

[2] V. W. Ajin and L. D. Kumar, 'Big data and clustering algorithms', in *International Conference on Research Advances in Integrated Navigation Systems, RAINS 2016*, Institute of Electrical and Electronics Engineers Inc., Dec. 2016. doi: 10.1109/RAINS.2016.7764405.

[3] M. A. Mahdi, K. M. Hosny, and I. Elhenawy, 'Scalable Clustering Algorithms for Big Data: A Review', *IEEE Access*, vol. 9, pp. 80015–80027, 2021, doi: 10.1109/ACCESS.2021.3084057.

[4] J. Nayak, B. Naik, and H. S. Behera, 'Fuzzy C-means (FCM) clustering algorithm: A decade review from 2000 to 2014', *Smart Innovation, Systems and Technologies*, vol. 32, pp. 133–149, 2015, doi: 10.1007/978-81-322-2208-8_14/COVER.

[5] J. Zubin, 'A technique for measuring like-mindedness', *J Abnorm Soc Psychol*, vol. 33, no. 4, pp. 508–516, Oct. 1938, doi: 10.1037/H0055441.

[6] V. N. Phu, N. D. Dat, V. T. Ngoc Tran, V. T. Ngoc Chau, and T. A. Nguyen, 'Fuzzy C-means for english sentiment classification in a distributed system', *Applied Intelligence*, vol. 46, no. 3, pp. 717–738, Apr. 2017, doi: 10.1007/S10489-016-0858-Z/METRICS.

[7] L. Zhu, F. L. Chung, and S. Wang, 'Generalized fuzzy C-means clustering algorithm with improved fuzzy partitions', *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 39, no. 3, pp. 578–591, 2009, doi: 10.1109/TSMCB.2008.2004818.

[8] M. N. Ahmed, S. M. Yamany, N. Mohamed, A. A. Farag, and T. Moriarty, ‘A modified fuzzy c-means algorithm for bias field estimation and segmentation of MRI data’, *IEEE Trans Med Imaging*, vol. 21, no. 3, pp. 193–199, 2002.

[9] T. Kwok, K. Smith, S. Lozano, and D. Taniar, ‘Parallel fuzzy c-means clustering for large data sets’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2400, pp. 365–374, 2002, doi: 10.1007/3-540-45706-2_48/COVER.

[10] M. Hashemzadeh, A. Golzari Oskouei, and N. Farajzadeh, ‘New fuzzy C-means clustering method based on feature-weight and cluster-weight learning’, *Appl Soft Comput*, vol. 78, pp. 324–345, May 2019, doi: 10.1016/J.ASOC.2019.02.038.

[11] J. MacQueen, ‘Classification and analysis of multivariate observations’, in *5th Berkeley Symp. Math. Statist. Probability*, University of California Los Angeles LA USA, 1967, pp. 281–297.

[12] H. Furuta, ‘Fuzzy logic and its contribution to reliability analysis’, *Reliability and Optimization of Structural Systems*, pp. 61–76, 1995, doi: 10.1007/978-0-387-34866-7_4.

[13] E. H. Ruspini, J. C. Bezdek, and J. M. Keller, ‘Fuzzy clustering: A historical perspective’, *IEEE Comput Intell Mag*, vol. 14, no. 1, pp. 45–55, Feb. 2019, doi: 10.1109/MCI.2018.2881643.

[14] J. C. Dunn, ‘A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters’, *Journal of Cybernetics*, vol. 3, no. 3, pp. 32–57, Jan. 1973, doi: 10.1080/01969727308546046.

[15] J. C. Bezdek, R. Ehrlich, and W. Full, ‘FCM: The fuzzy c-means clustering algorithm’, *Comput Geosci*, vol. 10, no. 2–3, pp. 191–203, Jan. 1984, doi: 10.1016/0098-3004(84)90020-7.

[16] S. Ghosh and S. K. Dubey, ‘Comparative analysis of k-means and fuzzy c-means algorithms’, *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 4, 2013.

[17] Y. Miao and Z.-Q. Liu, ‘On causal inference in fuzzy cognitive maps’, *IEEE Transactions on Fuzzy Systems*, vol. 8, no. 1, pp. 107–119, 2000, doi: 10.1109/91.824780.

[18] A. Stetco, X. J. Zeng, and J. Keane, ‘Fuzzy C-means++: Fuzzy C-means with effective seeding initialization’, *Expert Syst Appl*, vol. 42, no. 21, pp. 7541–7548, Nov. 2015, doi: 10.1016/J.ESWA.2015.05.014.

[19] J. Pérez-Ortega *et al.*, ‘Hybrid Fuzzy C-Means Clustering Algorithm Oriented to Big Data Realms’, *Axioms 2022, Vol. 11, Page 377*, vol. 11, no. 8, p. 377, Jul. 2022, doi: 10.3390/AXIOMS11080377.

[20] L. A. Zadeh, ‘Fuzzy sets’, *Information and Control*, vol. 8, no. 3, pp. 338–353, Jun. 1965, doi: 10.1016/S0019-9958(65)90241-X.

[21] S. Nascimento, B. Mirkin, and F. Moura-Pires, ‘Fuzzy clustering model of data and fuzzy c-means’, *IEEE International Conference on Fuzzy Systems*, vol. 1, pp. 302–307, 2000, doi: 10.1109/FUZZY.2000.838676.

[22] M. Gong, Y. Liang, J. Shi, W. Ma, and J. Ma, ‘Fuzzy C-means clustering with local information and kernel metric for image segmentation’, *IEEE Transactions on Image Processing*, vol. 22, no. 2, pp. 573–584, 2013, doi: 10.1109/TIP.2012.2219547.

[23] R. J. Hathaway and J. C. Bezdek, ‘Optimization of Clustering Criteria by Reformulation’, *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 2, pp. 241–245, 1995, doi: 10.1109/91.388178.

[24] D. Arthur and S. Vassilvitskii, ‘K-means++ the advantages of careful seeding’, in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007, pp. 1027–1035.

[25] J. Pérez-Ortega, N. N. Almanza-Ortega, and D. Romero, ‘Balancing effort and benefit of K-means clustering algorithms in Big Data realms’, *PLoS One*, vol. 13, no. 9, p. e0201874, Sep. 2018, doi: 10.1371/JOURNAL.PONE.0201874.

[26] O. Sakarya, ‘Applying fuzzy clustering method to color image segmentation’, *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015*, pp. 1049–1054, 2015, doi: 10.15439/2015F222.

[27] V. V. Vela-Rincón, D. Mújica-Vargas, and J. de Jesus Rubio, ‘Parallel hesitant fuzzy C-means algorithm to image segmentation’, *Signal Image Video Process*, vol. 16, no. 1, pp. 73–81, Feb. 2022, doi: 10.1007/S11760-021-01957-8/METRICS.

[28] A. A. M. Jamel and B. Akay, ‘A Survey and systematic categorization of parallel K-means and Fuzzy-c-Means algorithms’, 2019.

[29] Q. Zhang, Z. Chen, and Y. Leng, ‘Distributed fuzzy c-means algorithms for big sensor data based on cloud computing’, *International Journal of Sensor Networks*, vol. 18, no. 1–2, pp. 32–39, 2015, doi: 10.1504/IJSNET.2015.069871.

[30] J. Qin, W. Fu, H. Gao, and W. X. Zheng, ‘Distributed k-Means Algorithm and Fuzzy c-Means Algorithm for Sensor Networks Based on Multiagent Consensus Theory’, *IEEE Trans Cybern*, vol. 47, no. 3, pp. 772–783, Mar. 2017, doi: 10.1109/TCYB.2016.2526683.

[31] M. Al-Ayyoub, A. M. Abu-Dalo, Y. Jararweh, M. Jarrah, and M. Al Sa’D, ‘A GPU-based implementations of the fuzzy C-means algorithms for medical image segmentation’, *Journal of Supercomputing*, vol. 71, no. 8, pp. 3149–3162, Aug. 2015, doi: 10.1007/S11227-015-1431-Y/METRICS.

[32] N. A. Ali, B. Cherradi, A. El Abbassi, O. Bouattane, and M. Youssfi, ‘New parallel hybrid implementation of bias correction fuzzy C-means algorithm’, *Proceedings - 3rd International Conference on Advanced*

Technologies for Signal and Image Processing, ATSIP 2017, Oct. 2017, doi: 10.1109/ATSIP.2017.8075519.

[33] M. Al-Ayyoub, M. Al-andoli, Y. Jararweh, M. Smadi, and B. Gupta, 'Improving fuzzy C-mean-based community detection in social networks using dynamic parallelism', *Computers & Electrical Engineering*, vol. 74, pp. 533–546, Mar. 2019, doi: 10.1016/J.COMPELECENG.2018.01.003.

[34] S. AlZu'bi, M. Shehab, M. Al-Ayyoub, Y. Jararweh, and B. Gupta, 'Parallel implementation for 3D medical volume fuzzy segmentation', *Pattern Recognit Lett*, vol. 130, pp. 312–318, Feb. 2020, doi: 10.1016/J.PATREC.2018.07.026.

[35] P. Valsalan *et al.*, 'Knowledge based fuzzy c-means method for rapid brain tissues segmentation of magnetic resonance imaging scans with CUDA enabled GPU machine', *J Ambient Intell Humaniz Comput*, pp. 1–14, May 2020, doi: 10.1007/S12652-020-02132-6/METRICS.

[36] A. Manacero, E. Guariglia, T. A. de Souza, R. S. Lobato, and R. Spolon, 'Parallel fuzzy minimals on GPU', *Applied Sciences 2022, Vol. 12, Page 2385*, vol. 12, no. 5, p. 2385, Feb. 2022, doi: 10.3390/APP12052385.

[37] J. M. Cecilia, J. C. Cano, J. Morales-García, A. Llanes, and B. Imbernón, 'Evaluation of Clustering Algorithms on GPU-Based Edge Computing Platforms', *Sensors 2020, Vol. 20, Page 6335*, vol. 20, no. 21, p. 6335, Nov. 2020, doi: 10.3390/S20216335.

[38] J. M. Cebrian, B. Imbernón, J. Soto, and J. M. Cecilia, 'Evaluation of Clustering Algorithms on HPC Platforms', *Mathematics 2021, Vol. 9, Page 2156*, vol. 9, no. 17, p. 2156, Sep. 2021, doi: 10.3390/MATH9172156.

[39] N. A. Ali, A. El abbassi, and B. Cherradi, 'The performances of iterative type-2 fuzzy C-mean on GPU for image segmentation', *Journal of Supercomputing*, vol. 78, no. 2, pp. 1583–1601, Feb. 2022, doi: 10.1007/S11227-021-03928-9/METRICS.

[40] B. Liu, S. He, D. He, Y. Zhang, and M. Guizani, ‘A Spark-Based Parallel Fuzzy c -Means Segmentation Algorithm for Agricultural Image Big Data’, *IEEE Access*, vol. 7, pp. 42169–42180, 2019, doi: 10.1109/ACCESS.2019.2907573.

[41] Y. Ma and W. Cheng, ‘Optimization and Parallelization of Fuzzy Clustering Algorithm Based on the Improved Kmeans++ Clustering’, *IOP Conf Ser Mater Sci Eng*, vol. 768, no. 7, p. 072106, Mar. 2020, doi: 10.1088/1757-899X/768/7/072106.

[42] Q. Yu and Z. Ding, ‘An improved Fuzzy C-Means algorithm based on MapReduce’, *Proceedings - 2015 8th International Conference on BioMedical Engineering and Informatics, BMEI 2015*, pp. 634–638, Feb. 2016, doi: 10.1109/BMEI.2015.7401581.

[43] W. Dai, C. Yu, and Z. Jiang, ‘An Improved Hybrid Canopy-Fuzzy C-Means Clustering Algorithm Based on MapReduce Model’, *Journal of Computing Science and Engineering*, vol. 10, no. 1, pp. 1–8, Mar. 2016, doi: 10.5626/JCSE.2016.10.1.1.

[44] T. H. Sardar and Z. Ansari, ‘MapReduce-based Fuzzy C-means Algorithm for Distributed Document Clustering’, *Journal of The Institution of Engineers (India): Series B*, vol. 103, no. 1, pp. 131–142, Feb. 2022, doi: 10.1007/S40031-021-00651-0/METRICS.

[45] A. Almomany, A. Jarrah, and A. Al Assaf, ‘FCM Clustering Approach Optimization Using Parallel High-Speed Intel FPGA Technology’, *Journal of Electrical and Computer Engineering*, vol. 2022, p. 8260283, 2022, doi: 10.1155/2022/8260283.

[46] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[47] J. C. Zavala-Díaz, M. A. Cruz-Chávez, J. López-Calderón, J. A. Hernández-Aguilar, and M. E. Luna-Ortíz, ‘A Multi-Branch-and-Bound Binary Parallel Algorithm to Solve the Knapsack Problem 0–1 in a Multicore Cluster’,

Applied Sciences 2019, Vol. 9, Page 5368, vol. 9, no. 24, p. 5368, Dec. 2019, doi: 10.3390/APP9245368.