Algoritmo Meta-Heurístico para Clustering Particional de Datos basado en Global-Best Harmony Search, K-means y Restricted Growth Strings

Arnold Jair Jiménez Vargas

Director: PhD. Carlos Alberto Cobos Lozada
Co-Director: PhD. Martha Eliana Mendoza Becerra
Asesor: PhD. José Torres-Jiménez (CINVESTAV-Tamaulipas, México)

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas
Línea Investigación: Gestión de la Información -> Minería de Datos
Popayán, Junio de 2017

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPITULO 1

1. INTRODUCCIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

Hoy día, gracias a los avances tecnológicos tanto en recolección como almacenamiento de datos, es posible para los investigadores en diferentes áreas obtener grandes volúmenes de datos, ante lo cual surge la necesidad de contar con herramientas que permitan extraer la mayor cantidad de información útil [1] [2]. En el marco de la minería de datos, una de las tares más usadas para el análisis, es el agrupamiento o clustering de datos [3].

El Agrupamiento de datos es el proceso de dividir un conjunto de objetos en un número de grupos (clusters) previamente desconocidos minimizando la diferencia entre los objetos miembros de un mismo grupo y maximizando la diferencia entre los miembros de diferentes grupos [4]. El clustering es una de las tareas más desafiantes en el marco de la minería de datos y día a día se hacen nuevas propuestas para resolver esta tarea.

El agrupamiento de datos tiene muchas aplicaciones, por ejemplo: visión por computador (segmentación de imágenes), recuperación de información (agrupación de documentos web y de textos completos), biología (agrupamiento de genes), investigaciones de mercadeo (segmentación de clientes, mercados y predicción), entre otras [5] [6].

Los algoritmos de clustering pueden clasificarse en: jerárquicos, particionales, basado en densidad, basados en grillas, probabilísticos, espectrales y con restricciones [5]. Los algoritmos particionales son los más ampliamente usados y reportados en la literatura porque tiene un buen rendimiento y, además, hoy en día se pueden usar en aplicaciones de Clustering con grandes volúmenes de datos (big data Clustering) [7].

Existen muchos algoritmos para el clustering particional, de ellos el más conocido y estudiado es K-means que genera subconjuntos de datos agrupados alrededor de un centroide. K-means fue propuesto en 1957 en los laboratorios Bell [8] y se ha convertido en punto de partida para la creación de otras técnicas recientes, por su sencillez, facilidad de interpretación e implementación, capacidad de escalar y su capacidad para manejar flujos de datos (data stream) [6]. Esta técnica es propensa a mejoras debido a que en su forma original es 1) computacionalmente costoso, 2) los resultados dependen mucho de la selección inicial de los centroides (reportando un óptimo local y no el óptimo global), 3) requiere que

previamente se defina el número de grupos a generar, 4) es muy sensible al ruido, 5) sólo genera agrupaciones hiperesféricas o gausianas (no de otras formas) y 6) no define los atributos (características) de los datos que son más relevantes para el proceso de agrupación.

Entre las mejoras a K-means, en [1] se encuentra una mejora al proceso de reasignación de los datos a los grupos, en [3] una mejora al proceso de cálculo de los centroides a través de una heurística basada en el mejor centroide encontrado, en [9] una mejora al proceso de selección de centroides iniciales utilizando una distribución de probabilidad ponderada, en [10] un algoritmo que busca soluciones globales de clustering con selección de características y sin definir previamente el número de grupos, en [11] el uso de PCA para encontrar los centroides iniciales y una modificación a la re-asignación de los puntos a los grupos y en [12] un algoritmo de clustering llamado ModEx para seleccionar los centroides iniciales.

En esta propuesta se busca proponer un algoritmo que solucione los tres primeros problemas enunciados de K-means, ellos son: el costo computacional, el reporte de óptimos locales y la definición previa del número de grupos. Para resolver estos problemas se tuvieron en cuenta tres componentes principales, a saber: 1) Global-best Harmony Search [13], una meta-heurística competitiva en el estado del arte capaz de resolver problemas de optimización continuos, enteros y binarios, y que como todas las meta-heurísticas (Algoritmos Genéticos, algoritmos Meméticos, Evolución Diferencial, Optimización basado en Enjambre de Partículas, entre otros) está diseñado para buscar la solución óptima global de un problema; 2) Restricted Growth Strings (RGS) [14], como una estrategia de representación de una solución de agrupación que disminuye el espacio de búsqueda de las soluciones, hecho que permite encontrar mejores soluciones en menos tiempo de cómputo, y 3) Índices o criterios internos de calidad (como el criterio de información de Akaike, AIC) [15]_que permiten comparar soluciones de clustering de diferente o igual número de grupos para seleccionar la mejor alternativa de agrupación, lo que evitará que el usuario requiera definir previamente el número de grupos a obtener.

Teniendo en cuenta lo anterior, en la presente propuesta se buscó resolver la siguiente pregunta de investigación ¿Es posible obtener un algoritmo para clustering de datos basado en Global-best Harmony Search, K-means como optimizador local, Restricted Growth Strings y criterios de calidad internos como AIC, que sea competitivo en calidad con los reportados por otros algoritmos del estado del arte, pero con menor costo computacional y sin definir previamente el número de grupos (clustering automático)?

1.2 JUSTIFICACIÓN

Desde la perspectiva de investigación, los aportes de este trabajo de grado, se centran en la generación de nuevo conocimiento obtenido a partir de la evaluación del comportamiento del nuevo algoritmo propuesto para el clustering de datos, a

partir de conceptos que en su conjunto no se han reportado a la fecha en las bases de datos bibliográficas de IEEE, Springer, ACM y ScienceDirect. Adicionalmente, esta investigación puede ser usada como referencia en el desarrollo de diversas aplicaciones que se basan en el clustering de datos.

Desde la perspectiva de innovación, esta investigación propone un nuevo enfoque, uso de la Mejor Búsqueda Armónica Global (GBHS) con representación de soluciones basado en Restricted Growth Strings (RGS) e hibridado con k-means para la optimización local de las soluciones y de esta forma generar mejores soluciones al problema de clustering de datos, todo soportado en una función objetivo seleccionada de las que están reportadas en el estado del arte (índices de calidad internos) y que permite orientar la selección de la mejor solución para cada problema de clustering.

Este estudio es de tipo Exploratorio y Descriptivo y responde a preguntas como: ¿Es posible o no obtener mejoras en el clustering de datos con el algoritmo propuesto, frente a otros algoritmos ya reportados en el estado del arte y usando los mismos datasets de evaluación? y ¿Cuáles son los factores que hacen que este nuevo enfoque genere resultados con mayor o menor calidad, medidos con precisión, recuerdo y medida F?

1.3 OBJETIVOS

A continuación, se presentan los objetivos como fueron aprobados en el anteproyecto por parte del Consejo de Facultad, de la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca.

1.3.1 OBJETIVO GENERAL

Proponer un Algoritmo Meta-Heurístico para Clustering Particional basado en Global-best Harmony Search, K-means y Restricted Growth Strings que permita agrupar datos sin definir previamente el número de grupos.

1.3.2 OBJETIVOS ESPECÍFICOS

- Modelar un algoritmo de clustering de datos basado en la meta heurística de la mejor búsqueda armónica global como estrategia de optimización global, K-means como estrategia de optimización local, Restricted Growth Strings como esquema de representación de las soluciones y una función objetivo parametrizable.
- Definir la función objetivo para el algoritmo propuesto basado en índices internos de calidad reportados en el estado del arte como Bayesian Information Criterion, Akaike Information Criterion, la dispersión de las matrices y la covarianza de los grupos, entre otros.
- Evaluar el algoritmo propuesto utilizando medidas externas clásicas como precisión, recuerdo y medida F, en data sets clásicos del repositorio de la

Universidad de California-Irvine (UCI) y comparar su resultado frente a los obtenidos por otros métodos del estado del arte.

1.4 RESULTADOS OBTENIDOS

- El presente documento que detalla la motivación del proyecto y el estado del arte del mismo, el proceso de desarrollo llevado a cabo, en el que se incluye para cada una de las fases, los aportes significativos, los problemas encontrados, las alternativas planteadas y las soluciones seleccionadas. Se presenta el diseño detallado del algoritmo final junto con los resultados de su evaluación y comparación. Finalmente, las conclusiones del trabajo y las propuestas que el grupo de I+D espera llevar a cabo en el futuro próximo.
- Código fuente de la aplicación tipo consola implementada en Java y construida con Apache Maven, que realiza el clustering de los datasets de acuerdo a los parámetros configurados en un archivo json. El código fuente está además disponible en github (https://github.com/arnoldjair/trabajoGrado).
- Un artículo que resume los resultados de la investigación, en formato IEEE y que se espera se pueda publicar en evento o revista nacional.

1.5 ESTRUCTURA DE LA MONOGRAFÍA

A continuación, se describe de manera general el contenido y organización de la presente monografía.

CAPÍTULO 1: Introducción

Corresponde al presente capitulo, en el que se hace una introducción al tema de investigación, se presenta la pregunta de investigación que origino el trabajo, la justificación del problema, los objetivos (general y específicos) definidos al inicio del proyecto, un resumen de los resultados obtenidos y finalmente la organización de la monografía.

CAPÍTULO 2: Contexto teórico

En el segundo capítulo se presenta una descripción y explicación detallada de los conceptos (contexto teórico) utilizados en el presente proyecto de investigación, incluyendo: Cadenas de crecimiento restringido, el algoritmo de la mejor búsqueda armónica global, el algoritmo de k-means y los índices de calidad que permiten evaluar y comparar soluciones de clustering.

CAPÍTULO 3: Estado del arte

En el tercer capítulo se presentan las investigaciones más notables y relacionadas con el problema de investigación tratado en el presente proyecto.

CAPÍTULO 4: Enfoque Metodológico

En el cuarto capítulo se presenta la metodología seguida en el desarrollo el presente trabajo de investigación, así como el trabajo realizado, los problemas encontrados y las soluciones implementadas en cada una de las iteraciones desarrolladas.

CAPÍTULO 5: Propuesta

En este capítulo se hace una descripción detallada de la solución al problema de clustering particional de datos utilizando GBHS, K-means y RGS.

CAPÍTULO 6: Experimentación

En este capítulo se describe el proceso utilizado para la evaluación del algoritmo propuesto y los algoritmos del estado del arte utilizados para comparar su desempeño. Se muestra el proceso de afinamiento de los parámetros del algoritmo y finalmente los resultados obtenidos tanto por el algoritmo propuesto como por los del estado del arte. Finalmente, se realiza un análisis de los resultados obtenidos.

CAPÍTULO 7: Conclusiones y Trabajos a futuro

En este capítulo se presentan las conclusiones a las que se llegó después de culminar el proyecto y las mejoras que se pueden hacer al trabajo desarrollado.

CAPÍTULO 8: Bibliografía

Este capítulo contiene las referencias a los artículos, libros, páginas web y otros documentos consultados en la realización del proyecto.

CAPITULO 2

2. CONTEXTO TEÓRICO

2.1 INTRODUCCIÓN

En esta sección se presentan en forma general algunos conceptos fundamentales que se usan en minería de datos, en el contexto de clustering particional y meta-heurísticas y que ayudan a entender de una mejor forma el trabajo realizado.

2.2 CONJUNTO DE PARTICIONES (SET PARTITIONS)

Un conjunto de particiones [16] divide el conjunto N = {1,2,3,…,n} en k subconjuntos disjuntos no vacíos, llamados bloques, cuya unión es N. Por ejemplo, dado el conjunto N = {1,2,3}, se tienen los subconjuntos: {1,23}, {12,3}, {13,2}, {1,2,3} y {123}. El conjunto, {123}, tiene un k de uno, los siguientes {1,23}, {12,3}, {13,2} tienen un k de dos y el último {1,2,3} un k de tres, es decir, tres grupos cada uno de un elemento.

El número de particiones que se pueden conseguir a partir de un conjunto de N elementos se conocen como los números de Bell, siempre y cuando no se tengan en cuenta las particiones equivalentes, por ejemplo {1,23} es equivalente a {23,1}. Los números de Bell satisfacen la relación de recurrencia representada por la Ecuación (1).

$$B_{n+1} = \sum_{k=1}^{n} \binom{n}{k} B_k \qquad (1)$$

Donde $\binom{n}{k}$ en un coeficiente binomial [17].

2.3 RESTRICTED GROWTH STRINGS

Una cadena de crecimiento restringido o RGS [14] es una cadena $a[1 \ldots n]$ donde $a[i]$ es el bloque en el cual el elemento i ocurre. A continuación, en la Figura 1 se presentan los subconjuntos generados a partir de un conjunto N = {1,2,3,4} y su representación RGS. En dicha figura, el subconjunto {123.4} se representa como 0001 indicando que los elementos 1, 2 y 3 pertenecen al primer bloque (bloque 0) mientras que el elemento 4 pertenece al segundo bloque (bloque 1).

Estas cadenas están caracterizadas por la siguiente in-Ecuación (2) de crecimiento:

$$a[i+1] \leq 1 + \max(a[1], a[2], \ldots, a[i]) \quad (2)$$

Con $a[1] = 0$.

Donde el elemento i de la cadena no puede pertenecer a un grupo cuyo índice sea mayor en una unidad al índice mayor de los grupos de los elementos anteriores. Las RGS sirven entonces para representar el conjunto de particiones y debido a la restricción del crecimiento, los bloques (grupos) están ordenados en un orden canónico (normalizado) basado en el elemento más pequeño de cada bloque, asegurando que cadenas equivalentes como 0011 o {12.34} y 1100 o {34.12} no se consideren diferentes.

{1234}						
0000						
{123.4}	{124.3}	{134.2}	{1.234}	{12.34}	{13.24}	{14.23}
0001	0010	0100	01111	0011	0101	0110
{1.2.34}	{1.24.3}	{1.23.4}	{14.2.3}	{13.2.4}	{12.3.4}	
0122	0121	0112	0120	0102	0012	
{1.2.3.4}						
0123						

Figura 1: Representación RGS para conjunto N = {1,2,3,4}

2.4 ALGORITMOS META-HEURISTICOS

Las meta-heurísticas son “Métodos de solución que organizan una interacción entre procedimientos de mejora local y estrategias de alto nivel para crear un proceso capaz de escapar de óptimos locales y de realizar búsquedas robustas en un espacio de soluciones” (Adaptado de [18]). Ejemplos de este tipo de algoritmos son: el Simulated Annealing (Recocido Simulado) que se inspira en el proceso físico del templado de metales; los algoritmos genéticos que están basados en la teoría de la selección natural de Darwin según la cual las especies desarrollan características necesarias para sobrevivir en un entorno y luego las pasan a las siguientes generaciones; los algoritmos físicos buscan reproducir el comportamiento de fenómenos físicos, como la transformación de la energía o la propagación de sonido; Los algoritmos de inteligencia colectiva buscan reproducir el comportamiento de enjambres como los de abejas o de hormigas, cuyos agentes interactúan entre sí y su entorno siguiendo reglas simples pero que dotan al sistema de un comportamiento complejo, auto-organizado y descentralizado; y los algoritmos bio-inspirados que buscan reproducir el comportamiento de organismos biológicos, que conforman sistemas auto-organizados, distribuidos y adaptables, como lo es por ejemplo el sistema inmune.

2.5 LA BUSQUEDA ARMÓNICA

El algoritmo de la búsqueda armónica (Harmony Search, HS) [19], es un algoritmo meta-heurístico que simula el proceso de improvisación musical, en el cual los músicos buscan producir una armonía agradable determinada por el estándar estético auditivo [20]. Cuando un músico está improvisando, el realiza una de las siguientes acciones:

1. Toca alguna melodía conocida o aprendida anteriormente.
2. Toca algo parecido a la melodía anteriormente mencionada, ajustándola un poco al tono deseado.
3. Compone una nueva melodía basándose en sus conocimientos musicales a partir de nuevas notas seleccionadas aleatoriamente.

Estas tres acciones formalmente definidas en [19] corresponden a los componentes del algoritmo: Uso de la memoria armónica, ajuste de tono y aleatoriedad, respectivamente.

En la improvisación musical, si los conjuntos de notas tocadas son considerados una buena armonía, estas son guardadas en la memoria de cada músico, incrementando la posibilidad de hacer una buena armonía la próxima vez. Del mismo modo en el proceso de optimización en ingeniería, cada variable inicialmente toma valores aleatorios dentro del rango posible, formando un vector solución. Si los valores que conforman dicho vector son una buena solución, esta se almacena en la memoria, aumentando la posibilidad de encontrar mejores soluciones en la siguiente iteración.

HS ha sido aplicado en diversos problemas reales obteniendo resultados satisfactorios y sobre el cual se han realizado diversas mejoras, entre las cuales se destacan las siguientes: la búsqueda armónica mejorada (IHS, Improve Harmony Search) [21], la mejor búsqueda armónica global (GHS, Global-Best Harmony Search) [13] y la nueva búsqueda armónica global (NGHS) [22]. Estos nuevos algoritmos han tenido en cuenta para su desarrollo diversas técnicas de optimización existentes, entre ellas Particle Swarm Optimization (PSO) [23], mutación genética y el ajuste dinámico de parámetros, para disminuir el ruido en el algoritmo original, soportar mayor dimensionalidad y aumentar la precisión y la velocidad de convergencia a las soluciones, entre otras.

2.6 MEJOR BUSQUEDA ARMÓNICA GLOBAL

Global-best Harmony Search (GBHS) [13] es un algoritmo de optimización estocástica propuesto en el año 2008 por Mahamed G.H. Omran y Mehrdad Mahdavi, el cual hibrida la búsqueda armónica original con el concepto de inteligencia de enjambre propuesto en PSO, donde un enjambre de individuos (llamados partículas) vuela a través del espacio de búsqueda. Cada partícula representa un candidato a la solución del problema de optimización. La posición de una partícula está influenciada por la mejor posición visitada por sí misma (es

decir, su propia experiencia) y la posición de las mejores partículas en el enjambre (es decir, la experiencia de enjambre). GHS modifica el paso de ajuste del tono en HS de modo que la nueva armonía puede imitar a la mejor armonía en la memoria armónica. Este cambio permite a GHS trabajar eficientemente en problemas continuos y discretos. En general GHS es mejor que IHS y HS. El GBHS tiene los mismos pasos que IHS con la salvedad de la modificación del paso de improvisación de una nueva armonía.

Los parámetros del algoritmo son los siguientes:

- HMS: Tamaño de la memoria armónica. Está constituido por un conjunto de soluciones inicialmente generadas de manera aleatoria y ordenadas de acuerdo al valor del fitness. Esta memoria es utilizada para almacenar las mejores soluciones encontradas durante la ejecución el algoritmo.
- HMCR: Tasa de consideración de la memoria armónica. Este valor determina la probabilidad en cada iteración de generar un nuevo armónico a partir de los valores de la memoria armónica (Balance entre exploración y explotación del espacio de búsqueda).
- NI: Número de improvisaciones.
- PAR: Tasa de ajuste de tono. Cuando la nueva armonía se genera a partir de la memoria armónica (explotación), este valor determina si una determinada dimensión se obtendrá de un elemento cualquiera de la memoria armónica o de la mejor solución encontrada hasta el momento. Este valor es calculado a partir de los parámetros minPAR y maxPAR.

2.7 ALGORITMO KMEANS

Es el algoritmo de clustering de datos más sencillo y utilizado [24]. El algoritmo busca disminuir la suma del error cuadrado (Sum of Squared Error, SSE) cada vez que reconfigura los grupos. El algoritmo consta de los siguientes pasos básicos:

- Creación de los k clusters iniciales. Normalmente seleccionado en forma aleatoria del espacio de búsqueda, del dataset o con otra estrategia.
- Repetir.
    - Reasignación de puntos a ser incluidos en el cluster con el centroide más cercano, también conocido como el re-cálculo de membresías.
    - Cálculo de los centroides de cada cluster o actualización de centroides.
- Hasta que se cumpla un criterio de parada.

Hay diversos criterios de parada, entre ellos, que el proceso iterativo se detenga cuando los datos no cambien de cluster, o dichos cambios sean mínimos.

2.8 ÍNDICES DE CALIDAD

Estos índices son utilizados para determinar qué tan similares son los elementos de un mismo cluster y que tan disimilares lo son del resto [5][25]. Se dividen en índices internos y externos, donde los internos basan su cálculo en la compactibilidad y la separación, esto es, solo utilizan la información contenida en el dataset; los externos por su parte basan su cálculo en la información disponible sobre las clases y su asignación de los elementos del dataset. A continuación, se presentan los índices utilizados en el desarrollo del proyecto.

2.8.1 WGSS

La suma de cuadrados dentro del grupo o within-group sum of squares (WGSS), representada por la ecuación (3), mide la compactibilidad de un cluster.

$$\mathrm{WGSS} = \sum_{k=1}^{K} \sum_{i \in I_k} \left\| M_i^{\{k\}} - G^{\{k\}} \right\|^2 \qquad (3)$$

Donde $M_i^{\{k\}}$ y $G^{\{k\}}$ son un elemento y el centroide del k-ésimo cluster. Esta misma ecuación es utilizada para calcular a suma de errores cuadrados SSE (sum of squared errors).

2.8.2 BGSS

La suma de cuadrados entre grupos o between-group sum of squares (BGSS), representada por la ecuación (4), mide la separación entre los clusters.

$$\mathrm{BGSS} = \sum_{k=1}^{K} n_k \left\| G^{\{k\}} - G \right\|^2 \qquad (4)$$

Donde$G^{\{k\}}$ y n_k son el centroide y el número de elementos del k-ésimo cluster y G el centroide del dataset completo.

2.8.3 AKAIKE INFORMATION CRITERION (AIC)

El criterio de información de Akaike (Akaike information criterion) es utilizado en estadística para evaluar la calidad de un modelo. La idea principal consiste en obtener un modelo que ajuste bien los datos manteniendo una complejidad baja, para ello impone una penalización proporcional al número de parámetros del modelo. En clustering se utiliza la ecuación (5).

$$\mathrm{AIC} = \mathrm{argmin_k}[\mathrm{SSE(K)} + 2\mathrm{MK}] \qquad (5)$$

Donde M es el número de dimensiones (columnas) del dataset y K el número de grupos o clusters. SSE es una función de K, monótona decreciente, que toma el valor de cero cuando el número de grupos es igual al número de elementos del dataset (cada grupo estaría conformado por un solo elemento, por lo que el error sería cero). El SSE es un buen índice de calidad cuando el número de grupos es conocido, pero no así en el caso contrario, es por ello que se necesita del factor que penaliza el SSE con un valor proporcional al número de grupos.

2.8.4 BAYESIAN INFORMATION CRITERION (BIC)

El criterio de información Bayesiano sigue el mismo principio del AIC, aplicando una penalización proporcional a la complejidad del modelo. Para evaluar la calidad de una solución de clustering se utiliza la ecuación (6).

$$\mathrm{BIC} = \frac{1}{N} * \ln\left(\frac{N^K}{L^2}\right) \qquad (6)$$

Aquí L es el logaritmo de la verosimilitud (Log-Likelihood, a partir de ahora LL) y N el número de elementos del dataset.

Para calcular el LL se asume que los puntos del dataset siguen la misma distribución de probabilidad, en nuestro caso que siguen una distribución normal, la cual se expresa de acuerdo a la ecuación (7) para el caso univariante y a la ecuación (8) para el caso multivariante.

$$\mathrm{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{(2\pi\sigma^2)}} exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right) \qquad (7)$$

$$\mathrm{N}(\boldsymbol{X}|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}} exp\left(-\frac{1}{2}(\boldsymbol{X}-\mu)^T\Sigma^{-1}(\boldsymbol{X}-\mu)\right) \qquad (8)$$

2.8.5 CALINSKI-HARABASZ INDEX

Este índice relaciona BGSS y WGSS. Con ello se busca que los grupos sean compactos y que estén bien separados entre ellos, el valor de este índice aumenta a medida que se cumple con estas condiciones (se busca maximizar su valor). Se representa por la ecuación(9).

$$\mathrm{CHI} = \frac{(N-K)}{(K-1)}\frac{(BGSS)}{(WGSS)} \qquad (9)$$

2.8.6 DUNN INDEX

Este índice relaciona la distancia mínima entre puntos de diferentes grupos y la distancia máxima entre elementos del mismo grupo y al igual que el índice

anterior, el mayor valor obtenido se considera el adecuado. Se representa por la ecuación (10).

$$\mathrm{DI} = \frac{d_{min}}{d_{max}} \qquad (10)$$

Aquí d_{min} indica la distancia mínima entre dos puntos de diferentes grupos (separación mínima entre todos los pares de grupos) y d_{max} la distancia máxima entre dos puntos del mismo grupo (mayor diámetro de todos los grupos).

2.8.7 SILHOUETTE INDEX

Este índice se calcula de acuerdo a la ecuación (11).

$$\mathrm{SI} = \frac{1}{K}\sum_{k=1}^{K} s_k \qquad (11)$$

Donde s_k indica la media de los anchos de las siluetas de cada cluster k y se calcula de acuerdo a la ecuación (12).

$$\mathrm{s_k} = \frac{1}{n_k}\sum\nolimits_{i \in I_k} S(i) \qquad (12)$$

Aquí, $S(i)$ se calcula de acuerdo a la ecuación (13) e indica la amplitud de la silueta para el punto i.

$$\mathrm{S(i)} = \frac{b(i) - a(i)}{\max(a(i), b(i))} \qquad (13)$$

Donde $b(i)$ indica la distancia media entre el i-ésimo punto y los puntos de los otros clusters y $a(i)$ la distancia media entre el i-ésimo punto y los otros puntos del mismo cluster. Toma valores entre $[-1,1]$ indicando con valores cercanos a -1 que el i-ésimo punto es afectado por un grupo diferente al correcto y con valores cercanos a 1 es afectado por el grupo apropiado. En este índice el mayor valor obtenido se considera el adecuado.

2.8.8 NÚMERO DE INSTANCIAS CORRECTAMENTE CLASIFICADAS

Es un índice externo mide el número de registros del dataset que después del proceso de clustering quedan ubicados en el grupo correcto. Para calcularlo se requiere conocer tanto el número de grupos como la asignación de los registros en cada grupo.

Se construye una matriz de contingencia compuesta por $K + 1$ filas y $K' + 1$ columnas, donde K es el número de grupos encontrados tras el proceso de

clustering y K' el es número real de grupos. Cada celda C_{ij} con $0 < i < k$ y $0 < j < k'$ de esta matriz indica el número de elementos del cluster i que pertenecen a la clase cuyo índice es j. En la Figura 2 se muestra un ejemplo de una matriz de contingencia para un dataset de 20 elementos divididos en 4 grupos de igual número que después del proceso de clustering quedan organizados en 3 grupos. La fila $k + 1$ presenta la suma de cada columna, mientras que la columna $k' + 1$ presenta la suma de cada fila.

1	2	2	4	9
0	3	1	0	4
4	0	2	1	7
5	5	5	5	20

Figura 2: Matriz de contingencia

Tras la construcción de la matriz, se procede a etiquetar cada uno de los clusters encontrados de tal manera que el número de instancias correctamente clasificadas sea el máximo. Esto implica que necesariamente se deben evaluar todas las posibilidades y para ello se adaptó el algoritmo utilizado en weka. Dicho algoritmo consiste en la creación de un vector con la asignación de clases a clusters en un momento determinado y otro con la mejor asignación encontrada; y en la generación de las diferentes combinaciones posibles a través de un árbol sobre el cual se realiza una búsqueda en profundidad. En la Figura 3 se presenta un segmento del árbol de búsqueda generado para la matriz de contingencia de la Figura 2.

Cada uno de las hojas del árbol representa una asignación de clases a clúster y cada vez que el algoritmo genera una de estas asignaciones la compara con la mejor encontrada hasta ese momento y la actualiza si es el caso. Finalmente se selecciona aquella asignación que maximiza el número de instancias correctamente clasificadas, que en este ejemplo se da si se etiqueta el primer clúster con la clase 4, el segundo clúster con la clase 2 y el tercer clúster con la clase 1, que se representaría en el árbol como [4,2,1] y obtiene un NICC de 11.

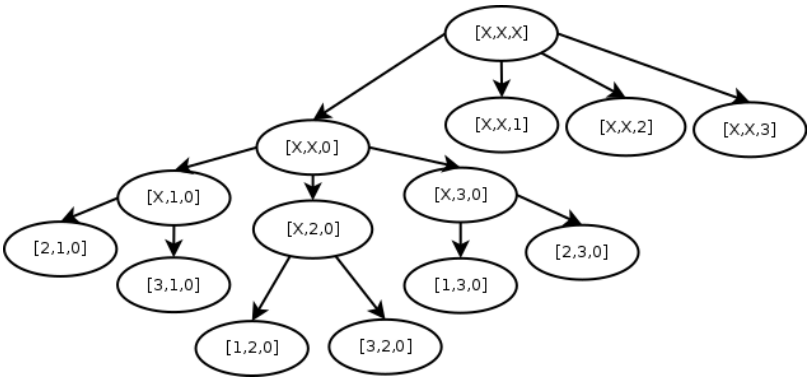

Figura 3: Segmento de árbol de búsqueda

2.8.9 ERROR

Este índice externo calcula el porcentaje de instancias incorrectamente clasificadas. Se calcula de acuerdo a la ecuación (14).

$$\text{ERROR} = \left(\frac{NIIC}{N}\right) * 100 \qquad (14)$$

Aquí $NIIC$ indica el número de instancias incorrectamente clasificadas y es el número total de registros del dataset menos el número de instancias correctamente clasificadas, que para el ejemplo anterior es 13.

2.8.10 RECUERDO

Este índice externo calcula el porcentaje de instancias correctamente clasificadas. Se calcula de acuerdo a la ecuación (15).

$$\text{RECUERDO} = \left(\frac{NICC}{N}\right) * 100 \qquad (15)$$

Aquí $NICC$ indica el número de instancias correctamente clasificadas.

3. ESTADO DEL ARTE

Debido a la desventaja principal del algoritmo K-means de depender de la selección de los centroides iniciales, la comunidad científica ha dedicado esfuerzos para mejorar su desempeño mejorando el proceso de selección de tales centroides y combinarlo con algoritmos meta-heurísticos ha reportado resultados prometedores. Estas integraciones han permitido además el desarrollo de algoritmos de clustering automático cuya principal característica es la de no necesitar que se defina previamente el número de grupos, sino que, durante su

ejecución, con la ayuda de diferentes índices de calidad, son capaces de determinar el valor de k más apropiado. A continuación, se presentan algunos trabajos relacionados con mejoras teóricas al algoritmo K-means, a la integración con diferentes meta-heurísticas y con mejoras en aplicaciones prácticas en diferentes campos.

3.1 CADENAS DE CRECIMIENTO RESTRINGIDO (RGS)

Restricted Growth String for Video-Type Classification [26] (2009): Este trabajo se centra en la clasificación de secuencias de videos en el campo del reconocimiento del modo de caminar de los humanos. Los autores definieron una metodología en la que un conjunto de características (condiciones de iluminación, objetos en movimiento a través de la secuencia, fondo de movimiento, cuadros por segundo y movimiento de la cámara) son extraídas manualmente de los videos e introducidas a un algoritmo RGS para ser agrupadas. Definieron una función para optimizar el crecimiento de las cadenas RGS para minimizar la distancia entre los elementos de un mismo cluster y la distancia entre clusters; y otra para determinar el centroide de los datos. Para comparar la metodología propuesta utilizaron dos bases de datos, cada una con 120 elementos y 5 características por cada uno e hicieron uso de K-means y SOM. Los resultados indican por un lado que la calidad de K-means decrece cuando crece la cantidad de clusters y que el porcentaje de clasificaciones correctas obtenidas con SOM es superior al de las obtenidas con RGS pero que el proceso de aprendizaje es más costoso.

3.2 ALGORITMO K-MEANS

K-means++: The Advantages of Careful Seeding [9] (2007): En este trabajo propusieron una mejora al proceso de selección de los centroides iniciales de los grupos. El primero de ellos denotado como c_1 seleccionado de manera aleatoria uniformemente del conjunto de datos inicial, los otros $k-1$ centros son seleccionados de manera aleatoria con probabilidad proporcional a la distancia al centro c_i más cercano. A partir de la selección de estos centros se procede con la ejecución del algoritmo K-means clásico. Los autores evaluaron con dos datasets reales y dos sintéticos, los resultados muestran que k-means++ ofrece un mejor rendimiento que el K-means clásico y que la calidad de los grupos encontrados es mejor.

Improving the Accuracy and Efficiency of the K-means Clustering Algorithm [1] (2009): En este trabajo buscaron mejorar el algoritmo K-means modificando el proceso de asignación de centroides iniciales y de reasignación de puntos. El proceso de asignación de centroides iniciales consiste en el cálculo de la distancia entre cada par de puntos del dataset original, la selección del par cuya distancia entre ellos sea mínima y la generación de un subconjunto A_i compuesto por tales puntos. Luego sigue la búsqueda del punto más cercano a A_i, que es agregado a este subconjunto y luego eliminado del dataset, repitiendo este paso hasta que el número de elementos de A_i alcanza un valor umbral. El proceso recién descrito es

repetido hasta que se obtienen k subconjuntos o clusters. Los centroides son generados promediando los valores de cada uno de los elementos de los subconjuntos A_i. El proceso de reasignación consiste en el cálculo de nuevos centroides y el cálculo de la distancia de cada punto respecto al nuevo centroide. Si la distancia al nuevo centroide es menor que la distancia al centroide anterior, entonces el punto está en el cluster correspondiente y no es necesario calcular la distancia a otros centroides, evitando el cálculo de otras distancias. Si la distancia es mayor entonces es posible asignar el punto a otro cluster, por lo que será necesario calcular la distancia a los otros centroides. El proceso es repetido hasta que, al igual que en el algoritmo K-means, se cumple con la condición de parada. Los resultados muestran una pequeña mejora respecto al tiempo de ejecución y precisión del algoritmo original.

A novel defect prediction method for web pages using K-means++ [27] (2015): En este trabajo aplicaron clustering en la detección de defectos en el código fuente de páginas web a través de una aplicación que implementa el algoritmo K-means++. En el estudio compararon el rendimiento de K-means++ con el de otros algoritmos utilizando la medida F, tiempo y tasa de error, se puede notar que K-means++ ofrece buenos resultados respecto al tiempo y tasa de error, pero no es muy apropiado cuando el tamaño del dataset es muy grande.

An Improved Clustering Algorithm for Text Mining: Multi-Cluster Spherical K-Means [28] (2016): En este trabajo propusieron un algoritmo para el clustering de documentos de texto basado en Spherical K-Means [29], que es una variación de K-Means utilizada para el clustering de textos en la que se utiliza la similitud (o disimilitud) de coseno y tanto los textos como los centroides son representados como un vectores de alta dimensión [30] cuya longitud está dada por la cantidad de términos diferentes que aparecen en cada texto. El algoritmo propuesto fue comparado con Spherical K-means (SKM) y Bisecting K-means (BKM) utilizando 3 datasets, dos de ellos en inglés y uno en turco, y las medidas de Pureza, Entropía e Información Mutua Normalizada. Los resultados de la evaluación sobre cada dataset muestran que el algoritmo propuesto ofrece una mejora significativa respecto a SKM y BKM.

3.3 ALGORITMOS METAHEURISTICOS

K-means Initialization Methods for Improving Clustering by Simulated Annealing [31] (2008): Los autores de este estudio propusieron el uso de PCA-Part (Enfoque determinista) [32] y K-means++ (Enfoque estocástico) como métodos de inicialización que ofrecen soluciones iniciales que son utilizadas como entrada al algoritmo meta-heurístico Simulated Annealing. Utilizaron en la fase de pruebas un total de 8 datasets obtenidos del repositorio de la Universidad de California-Irvine [33] y de acuerdo a los resultados el enfoque propuesto ofrece mejores resultados que los obtenidos al utilizar K-means y Simulated Annealing y a nivel estadístico la inicialización con PCA-Part y K-means++ de Simulated Annealing tiene un mejor desempeño que su inicialización con K-means.

Application of Bio-inspired Metaheuristics in the Data Clustering Problem [4] (2011): En este estudio extendieron un trabajo previo [34] en el que aplicaron Algoritmos Genéticos (Genetic Algorithms GA) y Optimización por Colonia de Hormigas (Ant Colony Optimization ACO) al problema del agrupamiento, agregando al conjunto de algoritmos uno llamado Sistemas Inmunes Artificiales (Artificial Immune Systems AIS) [35] a través de dos implementaciones llamadas CLONALG y opt-aiNet. De acuerdo a los autores GA y ACO son meta-heurísticas apropiadas para el problema de agrupamiento según lo realizado en el estudio previo. El documento incluye una extensa descripción de estos algoritmos y un estudio empírico de cada uno de ellos aplicado a 5 datasets obtenidos del repositorio de la Universidad de California-Irvine, Comparando el rendimiento de GA con MA (Memetic Algorithms) [36], ACO con y sin búsqueda local y CLONALG con opt-aiNet, los rendimientos de cada par de algoritmos fueron comparados utilizando el test de Wilcoxon disponible en R [37] concluyendo que la versión MAC y ACO con búsqueda local ofrece mejores resultados que la versión puramente heurística y que opt-aiNet ofrece a su vez mejores resultados que CLONALG.

Wavelet Neural Networks Initialization Using Hybridized Clustering and Harmony Search algorithm: Application in Epileptic Seizure Detection [38] (2013): En este trabajo combinaron K-Mean y Fuzzy C-Means con Harmony Search para encontrar los vectores de traslación iniciales de las redes neuronales Wavelet. Las redes neuronales Wavelet (WNN) son una variante de las redes neuronales artificiales y contienen un conjunto de parámetros que deben ser configurados durante el proceso de entrenamiento. En este estudio se centraron en el uso de algoritmos de clustering para determinar los vectores de traslación utilizados para ubicar los nodos interiores de la red. Los algoritmos KM, HKM, FCM y HFCM fueron puestos a prueba en un caso real de detección de ataques epilépticos utilizando validación cruzada con las medidas de sensibilidad, especificidad y precisión. Los resultados obtenidos muestran que los algoritmos hibridados ofrecen mejores resultados que los algoritmos base.

A survey on nature inspired metaheuristic algorithms for partitional clustering [8] (2014): Este trabajo comprende una revisión del estado del arte de los algoritmos meta-heurísticos inspirados en la naturaleza y empleados en el clustering particional a la fecha de realización del estudio (2013). Según los autores, estos algoritmos meta-heurísticos puede dividirse en evolutivos, físicos, de inteligencia colectiva, bio-inspirados y otros, además de dividirse de acuerdo a si utilizan una o varias funciones objetivo. El estudio comprende una corta revisión de diferentes enfoques basados en los tipos de algoritmos recién mencionados, de diferentes funciones objetivos utilizadas para cuantificar la calidad de las particiones basadas en la similitud de los patrones y de los algoritmos multi-objetivo.

Stock Market Prediction Using Clustering with Meta-Heuristic Approaches [2] (2015): En este trabajo utilizaron K-means, PSO-K-means [39], Firefly K-means

[40] y BAT K-means [41] para el clustering de datos del movimiento del índice bursátil NSE-NIFTY de la India y BSE-NIFTY de Bombay. El estudio incluye una comparación sobre un conjunto de datos y tablas que muestran agrupaciones en diferentes valores de K, a saber, 2, 5, y 10, los valores de Mean square quantization error y de sum of intra cluster distances, los cuales indican que Firefly K-means para este problema en concreto ofrece mejores resultados.

A Novel Artificial Bee Colony Based Clustering Algorithm for Categorical Data [42] (2015): En este trabajo propusieron un algoritmo llamado ABC-K-modes (Artificial Bee Colony clustering based on K-modes) basado en K-modes [43], que es un algoritmo basado en K-means que utiliza una medida de similitud de coincidencia simple y reemplaza la media por la moda en la definición de los centroides, lo que le permite trabajar con datos categóricos, y en Artificial Bee Colony [44] que está basado en la forma en la que un enjambre de abejas recolecta néctar. Para evaluar el algoritmo propuesto utilizaron 6 datasets obtenidos del repositorio de la Universidad de California-Irvine, la medida de exactitud de Yang [45] y el índice Rand [46] y lo compararon con los algoritmos K-modes, Fuzzy K-modes y Genetic K-modes. De acuerdo a los resultados, el algoritmo propuesto muestra un desempeño superior a los demás.

Automatic clustering using nature-inspired metaheuristics: A survey [47] (2016): Este trabajo comprende una revisión del estado del arte de los algoritmos meta-heurísticos inspirados en la naturaleza y empleados en el clustering automático de datos a la fecha de realización del estudio (2014). A diferencia del trabajo realizado en [8], en éste hacen una revisión de los algoritmos que no necesitan conocer a priori el número de grupos a obtener (Clustering automático). El estudio comprende una descripción de los conceptos básicos utilizados en el clustering automático, una revisión de meta-heurísticas de estado simple (evolucionan una única solución), de meta-heurísticas en las que una población de soluciones potenciales cooperan para optimizar una función objetivo (de objetivo simple) y de meta-heurísticas que optimizan diferentes funciones objetivo (de objetivo múltiple). El estudio concluye con un análisis de los algoritmos más relevantes, tendencias y conclusiones.

CAPITULO 3

4. ENFOQUE METODOLÓGICO

En esta sección se presenta el proceso de desarrollo llevado a cabo, incluyendo para cada una de las iteraciones los aportes significativos, los problemas encontrados, las alternativas planteadas y las soluciones seleccionadas.

El proyecto se desarrolló en 4 iteraciones siguiendo las etapas definidas en el Patrón de Investigación Iterativa propuesto por Pratt [48]. El patrón es iterativo e incremental y contempla las siguientes 4 etapas:

- Observación: Se hacen observaciones de campo, en el marco de problema.
- Identificación: Con las observaciones realizadas se identifica el problema.
- Desarrollo: Se desarrolla una solución al problema identificado.
- Prueba: Se realizan pruebas a la solución desarrollada.

Además, se incluyó una etapa adicional relacionada con la Documentación y Divulgación de Resultados, en la que se fue generando la monografía y el artículo científico con la información recolectada durante el desarrollo del proyecto.

4.1 PRIMERA ITERACIÓN

En la primera iteración se diseñó e implementó la lectura de los datasets (1), la representación de las soluciones con RGS (2) y diferentes índices internos de evaluación de la calidad de las soluciones de clustering (3), como se detalla a continuación.

4.1.1 DATASETS

Los datasets, obtenidos del repositorio de la UCI (University of Carolina at Irvine), se descargan como un archivo de valores separados por comas. Utilizando el editor de texto Kwrite se convierten a un formato similar al arff utilizado en weka, el cual tiene el formato mostrado en la Figura 4.

@name <nombre del dataset>	Indica el nombre del dataset.
@attribute <nombre del atributo> <tipo>	Indica un atributo del dataset. Los tipos soportados son double y class.
@data	Indica el inicio de los datos, los cuales están separados por tabulaciones.

Figura 4: Estructura de los datasets

A continuación, en la Figura 5 se presenta un segmento del dataset Iris.

```
@name       iris
@attribute  sepalLength  double
@attribute  sepalWidth   double
@attribute  petalLength  double
@attribute  petalWidth   double
@attribute  class  class

@data
5.1   3.5   1.4   0.2   Iris-setosa
4.9   3.0   1.4   0.2   Iris-setosa
```

Figura 5: Segmento del dataset iris

Cada dataset fue validado, primero, buscando las etiquetas y luego durante la lectura de los datos verificando que el número de datos leídos por cada registro corresponda al número de atributos definidos.

Una vez leído el dataset, cuyos datos son almacenados en una matriz de Objects, sus datos son normalizados utilizando la técnica min-max [49] representada por la Ecuación (16).

$$\acute{v} = \frac{(v - min)}{(max - min)} \qquad (16)$$

Este proceso de normalización se hace por cada columna o atributo del dataset, donde se busca su valor máximo y su valor mínimo, luego cada valor v es reemplazado por el resultado obtenido en la ecuación.

4.1.2 ALGORITMO RGS

Se encontró que las cadenas RGS se pueden generar en forma secuencial y en forma aleatoria.

Las RGS se pueden generar de manera secuencial de acuerdo al algoritmo descrito en [50], el cual asegura la unicidad y la restricción de crecimiento de las cadenas, pero no permite obtener una cadena cualquiera de la sucesión. Dado que el número de particiones crece muy rápidamente conforme lo hace el número de elementos tal y como se puede ver en [51], no es posible recrear el espacio de búsqueda y a partir de ahí seleccionar RGS para utilizarlas en K-Means. Por lo anterior, las RGS se deben generar de manera aleatoria, pudiendo darse los dos casos siguientes:

- Se puede generar un vector de n elementos, asegurando que el primer elemento es cero, y a partir de ahí en cada posición posterior del vector agregar un número aleatorio generado entre cero y el mayor de los números asignados previamente más uno sin sobrepasar el valor deseado de grupos.

- Se puede generar un vector de enteros aleatorios entre cero y el valor máximo de grupos deseados, el cual debe luego ser reordenado para que cumpla con la restricción de crecimiento.

Las RGS se constituyen en un modo de inicializar el algoritmo K-Means, cuya salida es una RGS indicando los grupos generados y la asignación de cada registro del dataset. Con esto se logra que se disminuya la visita repetida de soluciones.

4.1.3 ÍNDICES DE CALIDAD

Los índices internos se utilizan durante el proceso de clústering para determinar si una solución obtenida es mejor a las ya encontradas, los externos en cambio para determinar la bondad del resultado encontrado por el algoritmo.

En esta etapa se seleccionaron e implementaron los índices internos AIC, BIC, CHI, DI y SI y los índices externos Error y Recuerdo. Estos fueron seleccionados de acuerdo a los resultados presentados en [5].

Con BIC surgieron dos problemas: por un lado, el hecho de que hubiese grupos con un solo elemento por lo cual no es posible obtener una matriz de covarianzas; y por otro que la matriz de covarianzas no fuera invertible. Una variación de K-Means llamada X-Means [52] que consiste en la división iterativa de los grupos hasta cumplir con un criterio de parada (Iniciando con un único grupo que contiene a todos los elementos del dataset), utiliza el BIC para determinar qué particiones son las apropiadas. La ecuación del BIC utilizada en este estudio se presenta en la ecuación (17).

$$\mathrm{BIC} = \widehat{l_j}(D) + \frac{p_j}{2}\log(R) \qquad (17)$$

Aquí $\widehat{l_j}$ indica el LL, p_j el número de parámetros y R el número de elementos del dataset.

En X-Means no utilizan la matriz de covarianzas como se puede apreciar en la ecuación (18), en su lugar asumen que los grupos tienen una varianza constante $\widehat{\sigma^2}$ ˆ cuya estimación máximo-verosimil (Maximum Likelihood Estimate o MLE) se muestra en la ecuación (19).

$$l_j(\mathrm{D}) = \sum_{n=1}^{K}\left[R_n\left(log\frac{R_n}{R} - \frac{M}{2}log(2\pi\sigma^2)\right) - \frac{1}{2\sigma^2}\sum_{x_i\in D_n}\left\|x_i - \mu_{(i)}\right\|^2\right] \qquad (18)$$

$$\widehat{\sigma^2} = \frac{1}{R-K}\sum_i \left\|x_i - \mu_{(i)}\right\|^2 \qquad (19)$$

Esta es la base de la implementación de X-Means en weka [53], que a propósito tiene unos comentarios que indican que tal implementación no está hecha de acuerdo a lo indicando en el paper de X-Means. Debido a esto, se buscaron otras implementaciones y se encontró un desarrollo en github [54] cuyo autor afirma que el MLE de X-Means es incorrecto ya que los autores cometieron un error al momento de derivar la ecuación que da como resultado la ecuación (19). En github se encuentra un documento con la derivación de la ecuación y la forma correcta de encontrar el MLE, representada por la ecuación(20).

$$\widehat{\sigma^2} = \frac{1}{MR}\sum_i \left\| x_i - \mu_{(i)} \right\|^2 \qquad (20)$$

En [55] se encontró una implementación simplificada de BIC expresada por la ecuación (21).

$$\mathrm{BIC} = n * \ln\left(\frac{SSE}{n}\right) + k * \ln(n) \qquad (21)$$

En todo caso, con ninguna de las implementaciones de BIC revisadas se logró obtener resultados mejores, o al menos cercanos, a los obtenidos con AIC. Es por ello que se prescindió de este índice.

4.1.3.1 PRUEBAS

Para estas pruebas se utilizaron los datasets de Iris y Wine, cuyos datos se verificaron para asegurar que no hubiese datos erróneos (como caracteres o decimales con diferentes notaciones) o datos incompletos. Luego se transformaron con ayuda del editor de texto Kwrite al formato seleccionado.

Las pruebas se desarrollaron por un lado haciendo uso de la herramienta JUnit [56] que es un conjunto de herramientas para realizar pruebas unitarias en Java; y por otro del software estadístico R [57]. Las pruebas unitarias consisten en un conjunto de funciones que llaman a su vez a las funciones definidas en una clase y validan los resultados. En el caso de las RGS se implementaron las pruebas de las funciones encargadas de:

- Generar una partición aleatoria a partir de un número deseado de elementos y de grupos (n y k). El resultado de esta función es una partición cuya RGS contiene n elementos y a lo mucho k grupos, ambos valores pasados como parámetros.
- Generar una partición aleatoria a partir de un número deseado de elementos y de grupos (n y k) utilizando un objeto Random, todos estos valores pasados como parámetros. El resultado de esta función es similar al de la función anterior, salvo que diferentes ejecuciones bajo las mismas condiciones deben dar particiones iguales.

- Reprocesar una cadena que no necesariamente cumple con las restricciones de las RGS y retornar una partición que contenga la cadena como RGS valida.

Es necesario indicar que una partición es una clase que empaqueta una cadena RGS, un entero indicando el número de elementos y un entero indicando el número de grupos.

En el caso de la lectura de los dataset las pruebas no fueron automatizadas, en su lugar se verificó manualmente en modo depuración que los datos cargados fueran los mismos del archivo del dataset. Para verificar si los valores normalizados eran correctos, el dataset fue procesado en R y se verificó que el resultado entregado por el programa coincidiera con el resultado obtenido en R. Así mismo se realizaron las pruebas de los índices de calidad, siguiendo paso a paso cada uno de los algoritmos encargados de calcular tales índices verificando que no se dieran condiciones erróneas (como división por cero), fue aquí en donde se encontraron los problemas con la implementación de BIC y se decidió prescindir de su uso en las siguientes iteraciones.

4.2 SEGUNDA ITERACIÓN

En la segunda iteración se diseñó e implementó el algoritmo K-Means de manera tal que su entrada y salida estén representadas por RGS y se probaron diferentes criterios de parada.

4.2.1 ALGORITMO K-MEANS

El algoritmo K-means consiste en un proceso de reasignación de puntos al cluster más cercano el cual se repite hasta que se alcanza un criterio de parada, que generalmente es un número mínimo de reasignaciones, pero que también puede ser el valor alcanzado utilizando un índice.

Inicialmente se utilizó como criterio de parada un valor umbral de 0.1 calculado con la ecuación (22).

$$\mathrm{U} = \frac{R}{RA} \qquad (22)$$

Donde R es el número de reubicados en la iteración actual y RA el número de reubicados en la iteración anterior. Inicialmente RA es igual al número de elementos del dataset.

Este criterio tiene dos inconvenientes, por un lado, es posible que RA sea menor que R pudiendo U tomar valores mayores que 1, por otro lado, está la probabilidad de que existan puntos que estén oscilando entre grupos por lo cual nunca se

cumpliría el criterio de parada. Es por ello que se incluyó como segundo criterio un número máximo de iteraciones.

El punto de arranque del algoritmo lo constituyen un conjunto de puntos llamados centroides los cuales se pueden seleccionar al azar de entre todos los elementos del dataset o por medio de estrategias más elaboradas como ocurre en Kmeans++. En este proyecto, la configuración inicial está dada por las cadenas RGS generadas de manera aleatoria como se explicó en la iteración anterior. El algoritmo sigue los siguientes pasos:

1. Generación de la RGS.
2. Obtención de los clusters y cálculo de los centroides.
3. Ejecución del algoritmo K-means, reflejando las reasignaciones en la RGS, hasta que se cumple con el criterio de parada. En cada iteración se reprocesa la RGS para que cumpla con la definición.

Al ejecutar Kmeans se da el caso en el cual el nuevo valor de k es inferior al valor original, ya que en el proceso pueden desaparecer grupos, problema conocido en la literatura como el de las unidades muertas (dead units). Esto es posible debido a que los centroides no son elementos seleccionados del dataset, sino que son calculados (promediados) con los elementos de los grupos. Es por ello que se hace necesario el reprocesamiento de las RGS para obtener el nuevo número de elementos en cada iteración.

4.2.2 PRUEBAS

En este caso las pruebas consistieron en la ejecución paso a paso del algoritmo para verificar que no hubiera puntos que hicieran que el programa fallara, por ejemplo, quedando en un bucle infinito o en general en algún estado erróneo (pruebas de caja blanca). Tras verificar el correcto funcionamiento del algoritmo se implementaron pruebas unitarias para asegurar que cambios futuros en el código no afectaran los resultados de k-means.

Fue aquí cuando se descubrió que la ecuación utilizada para calcular el umbral (ecuación (22)) en algunas ocasiones entregaba valores mayores que 1 y que el número de iteraciones del algoritmo no alcazaba valores mayores que 30. Con estos resultados se decidió revisar nuevamente la literatura asociada a K-means en busca de otros criterios de parada.

4.3 TERCERA ITERACIÓN

En la tercera iteración se implementó el algoritmo GBHS y se integró K-means como optimizador local y RGS como representación de las soluciones.

4.3.1 ALGORITMO GKR REGISTROS

El algoritmo GBHS sigue los siguientes pasos (Adaptado de [13]):

1. Inicializar el problema y los parámetros.
2. Inicializar la memoria armónica.
3. Improvisar una nueva armonía.
4. Actualizar la memoria armónica.
5. Verificar el criterio de parada.

Para integrar K-means a GBHS, se agrega un paso nuevo entre los pasos 3 y 4 en el cual la nueva armonía es optimizada utilizando K-means. Este proceso de optimización se realiza si un número aleatorio uniforme generado entre 0 y 1 es menor que un parámetro que se denominó PO (probabilidad de optimización). De esta manera quedaron integrados GBHS, K-means y RGS.

Una vez hecha esta integración, se pensó en la posibilidad de no solo trabajar sobre los registros de las RGS uno a uno, sino en hacer asignaciones ya sea de grupos completos o de centroides en el paso de improvisación de una nueva armonía. Así, se definió un algoritmo para trabajar sobre grupos y otro para trabajar sobre centroides, los cuales se presentan a continuación.

4.3.2 ALGORITMO GKR GRUPOS

En esta versión del algoritmo se tuvo en cuenta que no todos los elementos de la memoria armónica tienen el mismo número de grupos, así que se definió una función para mezclar dos RGS cuyo propósito es combinar el i-ésimo grupo de la RGS1 con la RGS2, por esto, cuando se debe fijar un grupo de la nueva armonía escogiéndolo de un elemento de la memoria, de una armonía existente se escoge de manera aleatoria un índice de grupo y este se pasa como parámetro a la función junto a la armonía seleccionada y la nueva. También se definió una función para fijar aquellos valores de la nueva armonía que pudieran quedar vacíos fijándoles el valor de k. Previamente todos los elementos de la nueva RGS deben ser fijados en -1 indicando que está vacía.

4.3.3 ALGORITMO GKR CENTROIDES

En esta versión del algoritmo, las armonías se generan combinando centroides de la memoria armónica o generándolos de manera aleatoria. En este caso la nueva armonía está compuesta por un vector de centroides (representación de longitud variable para la armonía) y se invoca a una función que a partir del vector generado y el dataset retorna la RGS que lo representa, la cual recorre el dataset y busca el centroide más cercano a cada elemento cuyo índice se fija en la RGS.

4.3.4 PRUEBAS

Los algoritmos fueron puestos a prueba utilizando los 5 datasets seleccionados del repositorio de la UCI (Glass, Iris, Sonar, WDBC y Wine). Cada uno de ellos (Registros, Grupos y Centroides) fue ejecutado paso a paso para verificar que las nuevas armonías se estuvieran generando de manera apropiada, los cuales de acuerdo al porcentaje de optimización serían utilizados como puntos de partida de K-means. Se evaluó que los algoritmos no pudieran caer en estados erróneos y que la salida entregada por cada uno representara adecuadamente los resultados esperados. Durante la ejecución de los algoritmos se generó un archivo en el que se iba imprimiendo los resultados parciales, el cual incluye la memoria armónica inicial, la memoria armónica tras un proceso de optimización con K-means y en cada iteración la nueva armonía generada y la memoria armónica. Con esto se observó que los algoritmos estaban convergiendo de manera prematura a una solución puesto que no había mucha variabilidad en la memoria armónica (como un músico que solo es capaz de crear nuevas melodías con muy pocas variaciones entre ellas). También se observó que la memoria armónica inicial tras un proceso de optimización con K-means podía contener armonías (soluciones) iguales. Para la siguiente iteración se dispuso buscar una solución para estas dos situaciones.

4.4 CUARTA ITERACIÓN

En esta iteración se modificó la arquitectura de la aplicación para permitir la inclusión de nuevas características en trabajos futuros que facilitan además la compresión del código fuente, entre las que se tiene la configuración de los parámetros de entrada con un archivo json, el uso de hilos para aprovechar el uso de los recursos del equipo, la definición de interfaces para la inclusión de otras implementaciones de k-means, ghbs, distancias, filtros, entre otros.

4.4.1 PARÁMETROS DE ENTRADA

Los parámetros de entrada se definen en un archivo json con los siguientes campos:

- minPar: Par mínimo, valor real entre 0 y maxPar.
- maxPar, Par máximo, valor real entre minPar y 1.
- hmcr: Harmony memory consideration rate, valor real entre 0 y 1.
- hms: Harmony memory size, valor entero mayor que 1, valor recomendado por los autores de GBHS es entre 5 y 20.
- nExp: número de experimentos, valor entero mayor que 1 y recomendado en 30 para que los reportes cumplan con el teorema del límite central.
- po: Porcentaje de optimización, valor real entre 0 y 1.
- seed: Semilla aleatoria, valor entero positivo o cero.
- nIt: Número de iteraciones, valor entero mayor que 1.

- maxK: valor entero de k máximo.
- maxKMeans: Número de iteraciones máximo de k-means que corresponde a un número entero mayor que 1.
- threads: Valor entero que define el número de hilos que se ejecutarán en paralelo.
- datasets: Vector de cadenas con los nombres de los datasets a utilizar. Tales datasets deben estar en la carpeta determinada por el parámetro datasetsPath.
- algorithms: Vector de cadenas con los nombres de las versiones del GBHS.
- distances: Vector de cadenas con los nombres de las distancias.
- objectiveFunctions: Vector de cadenas con los nombres de las funciones objetivo que se van a evaluar.
- datasetsPath: Ruta absoluta a la carpeta con los datasets (los cuales deben estar en formato json)
- kmeansAlgorithm: Vector de cadena con los nombres de las versiones de k-means.
- initialization: Método de inicialización.
- normalize: Valor booleano que indica si los datasets deben ser normalizados.
- filters: Vector de objetos, cuyos campos son filter indicando el nombre del filtro y value indicando el valor.
- fixedK: Valor booleano que indica si el algoritmo debe trabajar con valor de k fijo, el cual debe estar presente en el archivo del dataset.
- log: Valor booleano que indica si se debe registrar en archivos individuales los valores de la memoria armónica en cada ejecución del algoritmo, esto con el objetivo de hacer un análisis detallado de la evolución de las soluciones.
- tuneUp: Valor booleano que indica si se debe ejecutar el proceso de afinación de los parámetros del algoritmo.
- tuneUpIt: Valor entero que determina el número de veces que se debe ejecutar el proceso de afinación de los parámetros del algoritmo.

En la Figura 6 se muestra un ejemplo del archivo con los parámetros de entrada.

```
{
        "minPar": 0.01,
        "maxPar": 0.99,
        "hmcr": 0.95,
        "hms": 30,
        "nExp": 10,
        "po": 1,
        "seed": 10,
        "nIt": 1000,
        "maxK": 10,
        "maxKMeans": 100,
        "threads": 4,
        "datasets": ["iris", "glass", "sonar", "wine", "wdbc"],
        "algorithms": [ "records"],
```

```
        "distances": ["euclidean", "manhattan"],
        "objectiveFunctions": ["aic", "chi"],
        "datasetsPath": "/home/equipo/datasets",
        "kmeansAlgorithm": "basic",
        "initialization": "random",
        "normalize": true,
        "filters": [{"filter": "stdDev", "value": 0.27}],
        "fixedK": false,
        "log": false
}
```

Figura 6: Ejemplo de archivo de parámetros en json

4.4.2 K-MEANS

Hasta el momento el criterio de parada de k-means estaba determinado por el número máximo de iteraciones (inicialmente 100) y por la relación entre el número de registros reasignados en determinada iteración y los reasignados en la iteración predecesora, pero se verificó que el promedio de iteraciones para que k-means converja a un óptimo local es 6.3 ± 2.5, así que se cambió el criterio de parada determinado por el umbral calculado según la ecuación (22) en lugar de detener la ejecución cuando no se hacen reasignaciones y el número máximo de iteraciones se fijó en 30 (aunque este valor es configurable en el archivo de parámetros de entrada del programa). Por otro lado, se implementó otra versión de k-means en la que el criterio de parada es, además del número de iteraciones, el que en determinada iteración el valor de una función objetivo no mejore respecto al valor obtenido en la iteración anterior. Aunque es posible implementar el k-means de manera tal que reciba como parámetro el criterio de parada, se dejaron como implementaciones aparte para que la estructura del proyecto quedara lista para agregar otras implementaciones de k-means en futuras mejoras (para ello se definió una interfaz (contrato software) la cual es implementada por ambas versiones de k-means).

4.4.3 GENERACIÓN DE PARTICIONES ALEATORIAS

Se incluyó el método de inicialización K-means++ para la generación de particiones aleatorias, la idea es utilizarlo cuando se genera la memoria armónica inicial. Se comprobó que en esta investigación no hay mejora entre la generación de la memoria armónica de manera aleatoria y la propuesta por este método.

4.4.4 SELECCIÓN DE CARACTERÍSTICAS

Se implementó un filtro que retira aquellos atributos del dataset cuya desviación estándar es menor que un valor configurado en el archivo de parámetros de entrada. Esta no es una característica propia de los algoritmos de clustering propuestos, sino que consiste en una etapa opcional de preprocesamiento de los dataset.

4.4.5 FUNCIÓN DE DISTANCIA

Se modificó la aplicación para que trabaje con diferentes funciones de distancia (euclidiana y manhattan), las cuales se pueden seleccionar en el archivo de parámetros de entrada y se pueden utilizar tantas como se tengan implementadas.

4.4.6 TAREAS

Se definió una estructura de datos para la ejecución de hilos, la cual consta de los parámetros de entrada con la salvedad de que cada tarea tiene solo un elemento de los vectores y se crean tantas tareas como combinaciones de elementos en los vectores, así por ejemplo se tiene una tarea que contiene el dataset Iris, el algoritmo gbhs-centroides, distancia euclidiana y la función objetivo AIC, con los demás parámetros definidos en el archivo json. En la implementación actual se pueden crear hasta 60 tareas dado que se trabajó con 5 datasets, 3 versiones de gbhs, 2 funciones de distancia y 2 funciones objetivo. Estas tareas son utilizadas por la clase encargada de la ejecución del algoritmo de clustering, estas clases implementan la interfaz Callable y son administradas por un ThreadPoolExecutor, ambas disponibles en java.

4.4.7 DATASETS

El formato de los datasets se cambió a json con el propósito de aprovechar las bondades que ofrece, como el que haya muchas herramientas para diferentes lenguajes de programación para trabajar con este formato y el que facilite la interoperabilidad con otros sistemas, esto se hizo pensado en los trabajos futuros en los que se pueda por ejemplo incluir aplicaciones web y bases de datos nosql [58].

4.4.8 AFINAMIENTO

Se implementaron dos afinadores, el primero de ellos basado en GBHS y el segundo en selección de parámetros de manera aleatoria.

En el afinador las armonías o soluciones están compuestos por los campos minPar, maxPar, hmcr, po, ER (error), hms, algoritmo GBHS (centroides, grupos y registros) y función objetivo. El objetivo es minimizar el valor del error y para ello por cada armonía generada se ejecuta el algoritmo GBHS 30 veces sobre cada dataset y se calcula el promedio del error de todas estas ejecuciones.

4.4.8.1 AFINADOR GBHS

En esta versión del afinador se generan HMS (harmony memory size del afinador) armonías que constituyen la memora armónica inicial, calculando para cada uno como se indicó anteriormente el valor del error asociado y luego se realiza el proceso de improvisación (en el que se configuraron 1000 iteraciones). El principal

problema de este afinador es que es muy costoso en tiempo de ejecución, llegando a tomar en promedio 4 días el proceso de afinación con los datasets Iris, Glass, Sonar, WDBC y Wine.

4.4.8.2 AFINADOR ALEATORIO

En esta versión del afinador se lee un archivo de parámetros en formato json similar al utilizado para el proceso de clústering, pero aquí se define adicionalmente el número de iteraciones del afinador. En cada iteración los valores de minPar, maxPar, hmcr, po, hms, maxK, maxKMeans y nIt son generados de manera aleatoria y se procede a crear la lista de tareas como se indicó anteriormente. El resultado de cada tarea es almacenado en un archivo de valores separados por comas indicando en cada fila los parámetros utilizados y el valor del error. La principal ventaja de este afinador respecto a la versión GBHS radica en la posibilidad de gestionar el tiempo que se dedica al proceso de afinación, pudiendo ejecutarse una iteración cada que se tenga disponibilidad, luego estos resultados pueden utilizarse para determinar el mejor conjunto de parámetros de entrada.

4.4.9 CONVERGENCIA PREMATURA

La convergencia prematura ocurre cuando hay poca variabilidad en la memoria armónica. Para solucionar este problema, se implementó una función para regenerar la memoria armónica en caso de que esta sea muy similar (poca diversidad), para ello se calcula la media y la desviación estándar de valor del fitness de las armonías de la memoria armónica y si el valor de la desviación es menor o igual al 5% del valor de la media, entonces se descartan todos los elementos a excepción de los dos mejores y se generan nuevas armonías de manera aleatoria. Si la memoria es regenerada 10 veces, entonces se da por terminado el proceso de clustering, de esta forma, se tiene que los criterios de parada de los algoritmos gbhs están dados por el número máximo de iteraciones y por el número de regeneraciones de la memoria armónica.

4.4.10 ARMONÍAS REPETIDAS

Se implementó una función que determina si una nueva armonía ya existe en la memoria. En las tres versiones del algoritmo de clustering, si la nueva armonía que se crea en el proceso de improvisación ya existe entonces la iteración se repite. Cuando la nueva armonía existe en el proceso de inicialización de la memoria armónica, entonces se descarta, pero el contador de iteraciones no se decrementa, así que al final se obtiene una memoria cuyo tamaño es menor o igual que el parámetro HMS. Los algoritmos, entonces, se modificaron para trabajar con una memoria armónica de tamaño variable y al generar una nueva armonía si el tamaño de la memoria es menor que HMS, entonces se agrega a la memoria, en caso contrario se sigue la lógica de reemplazar al peor (solución de menor calidad en la memoria).

5. ALGORITMO PROPUESTO

En esta sección se presentan los algoritmos propuestos que surgen de la hibridación de la Mejor Búsqueda Armónica Global (Global-Best Harmony Search) en sus tres versiones (registros, centroides y grupos), K-means y RGS como esquema de representación base o complementario.

5.1 REPRESENTACIÓN DE LAS SOLUCIONES

Cada solución (o armonía) consiste en un vector de $n + 1$ elementos de tipo numérico, donde n es el número de atributos del dataset (dimensionalidad) y el campo adicional está reservado para el valor de fitness. Debido a que los primeros elementos de una solución son de tipo entero y el último de tipo real se generaliza entonces como numérico y se definió de esta manera para evitar dependencia a algún paradigma o lenguaje de programación en particular.

5.2 ALGORITMO GBHS BASE

En la Figura 7 se presenta el pseudocódigo del algoritmo GBHS base.

Algoritmo GBHS

```
1: function GBHS(HMS, HMCR, NI, minPAR, maxPAR, N,
       ↪ OF)
2:     HM ← generateHarmonyMemory(HMS, OF)
3:     harmony ← double[N + 1]
4:     for cIt = 0 to NI − 1 do
5:         PAR ← MinPAR + (MaxPAR − MinPAR) ∗ (cIt/NI)
6:         for i = 0 to N − 1 do
7:             if U(0, 1) ≤ HMCR then
8:                 pos ← U(0, HMS − 1)
9:                 harmony[i] ← HM[pos][i]
10:                if U(0, 1) ≤ PAR then
11:                    pos ← U(0, N − 1)
12:                    harmony[i] ← HM[0][pos]
13:                end if
14:            else
15:                harmony[i] ← LB_i + r(UB_i − LB_i)
16:            end if
17:        end for
18:        HM ← replaceTheWorst(HM, harmony, OF)
19:    end for
20:    return HM[0]
21: end function
```

Figura 7: Algoritmo GBHS-Base

Inicialmente se genera la memoria armónica (línea 2), que está compuesta por HMS armonías que a su vez contienen un vector de números de coma flotante, enteros o binarios de acuerdo al problema de optimización, cuyo tamaño se fija de acuerdo al número de valores a optimizar y el valor del fitness. Esta lista de armonías (memoria armónica) se ordena de manera ascendente o descendente dependiendo de si el problema de optimización requiere maximizar o minimizar el valor de una función objetivo.

De ahí sigue la ejecución de NI iteraciones (líneas 4 a19) donde cada iteración consiste en un ciclo en el que se va generando cada uno de los valores de la nueva armonía y en el reemplazo de la peor armonía de la memoria armónica si la nueva es mejor. Cada valor de la armonía se genera de acuerdo a las siguientes reglas:

- Si un número aleatorio generado de manera uniforme entre cero y uno es menor o igual que HMCR (línea 7), entonces el valor se selecciona de una armonía de la memoria (línea 8 y 9), en caso contrario se genera de manera aleatoria tomando un valor entre el valor mínimo (LB) y el valor máximo (UB) del rango de búsqueda de la variable (línea 15).
- Si el valor debe ser seleccionado de una armonía de la memoria, entonces la armonía corresponde a la mejor si el valor de un número aleatorio generado de manera uniforme entre cero y uno es menor o igual que el valor de PAR (línea 10), cuyo valor se calcula en cada iteración utilizando los parámetros minPar y maxPar (línea 5), o en caso contrario, a una seleccionada de manera aleatoria de la memoria.

Cuando una nueva armonía reemplaza al peor de la memoria (línea 18), ésta se reordena nuevamente de manera ascendente o descendente según sea el caso. Finalmente, el resultado es la mejor armonía de la memoria. En este proceso de reemplazo se calcula el valor de fitness de la nueva armonía y se compara con el fitness de la peor solución en la memoria y la función objetivo debe indicar si maximiza o minimiza para poder hacer la comparación.

5.3 ALGORITMO K-MEANS

En la Figura 8 se presenta el pseudocódigo del algoritmo K-means. K-means consiste de un proceso iterativo que se ejecuta hasta que se alcanza un número máximo de iteraciones (maxKMeans) o mientras la relación entre los elementos reubicados en una iteración (reallocated) y los reubicados en una iteración anterior (prevReallocated) aquí llamada currPercent sea mayor que el parámetro pKMeans (Línea 29). En cada iteración se fija en cero el número de reubicados (línea 6), se recupera el número de grupos en la rgs (lo cual se representa con la función countGroups en la línea 7), se calculan los centroides que al igual que los elementos del dataset son vectores cuyo tamaño es igual a la dimensión del dataset (lo cual se representa con la función calculateCentroids en la línea 8) y se realiza el proceso de reasignación (cálculo de membresías, líneas 9 a 24).

Algoritmo KMeans

```
1: function KMEANS(rgs, maxKMeans, pKMeans, distance,
       ↪ dataset)
2:     N ← dataset.dimension
3:     currIt ← 0
4:     rgs : numeric[N + 1]
5:     do
6:         reallocated ← 0
7:         k ← countGroups(rgs)
8:         centroids ← calculateCentroids(rgs)
9:         for i = 0 to N − 1 do
10:            record ← dataset.getRecord(i)
11:            index ← rgs[i]
12:            dist ← distance(record, centroids[index])
13:            for j = 0 to k − 1 do
14:                currDist ← distance(record, centroids[j])
15:                if currDist < dist then
16:                    index ← j
17:                    dist ← currDist
18:                end if
19:            end for
20:            if index ≠ rgs[i] then
21:                reallocated + +
22:                rgs[i] ← index
23:            end if
24:        end for
25:        currPercent ← reallocated/prevReallocated
26:        prevReallocated ← reallocated
27:        rgs ← reprocessRGS(rgs)
28:        currIt + +
29:    while (currPercent > pKMeans)
       ↪ and (currIt < maxKMeans)
30:    return rgs
31: end function
```

Figura 8: Algoritmo K-means

El proceso de reasignación consiste en un ciclo en el que se recorre uno a uno los elementos del dataset (ciclo en la línea 9), se selecciona el cluster más cercano al elemento actual (líneas 13 a 18) y se fija el identificador de dicho cluster (número entero en el rango [0, k-1]) a la rgs en la posición dada por el valor del ciclo actual

(i). Si el valor de la posición de la rgs cambia, entonces se incrementa en uno el valor de los reasignados para esa iteración (línea 21).

Finalizado el proceso de reasignación, se actualizan los valores de currPercent y prevReallocated (líneas 25 y 26) y se reprocesa la rgs para que cumpla con la condición de crecimiento restringido y para poder obtener un número válido de grupos en la siguiente iteración si es el caso (línea 27).

5.4 ALGORITMO GKR-REGISTROS

En la Figura 9 se presenta el pseudocódigo del algoritmo GKR-Registros. Este algoritmo hereda de GBHS los pasos generales, a saber: generación inicial de la memoria armónica, el proceso de improvisación y el de reemplazo por el peor de la memoria.

En GKR la memoria armónica está constituida por RGS así que, en la función generadora, representada en este caso por generateHarmonyMemory (línea 2), se crean como máximo HMS cadenas aleatorias cada una con su respectivo valor de fitness, esto debido a que no se fuerza a la aplicación a que reemplace aquellas cadenas repetidas que son descartadas. Luego la memoria armónica se optimiza (línea 3), lo cual consiste en el recorrido de todos sus elementos y si un valor aleatorio uniforme entre 0 y 1 generado es menor que PO, entonces el elemento actual en el ciclo es procesado con K-means.

En el proceso de improvisación de una nueva armonía la diferencia consiste en que al generar uno de sus elementos de manera aleatoria, este se obtiene del intervalo [0, k-1] (línea 18), donde el valor de K es obtenido con una función que selecciona este valor aplicando las mismas reglas del proceso de improvisación del GBHS (línea 8), esto es, con HMCR y PAR se determina si el valor de k se obtiene de alguna armonía de la memoria o de manera aleatoria.

Tras la generación de una nueva armonía, como se puede ver en la línea 22, si un valor aleatorio generado uniformemente entre 0 y 1 es menor que PO, entonces este se optimiza con K-Means. Esto corresponde a la fase de explotación del espacio de búsqueda, que consiste en la obtención de una solución óptima local en el vecindario de la obtenida con el algoritmo GBHS, con el cual se lleva a cabo el proceso de exploración del espacio de búsqueda.

Algoritmo GKR

```
1: function GKR(HMS, HMCR, NI, minPAR, maxPAR, PO,
       ↪ maxKMeans, pKMeans, distance, dataset, OF, maxK)
2:     HM ← generateHarmonyMemory(HMS, OF, maxK,
       ↪ dataset, distance)
3:     HM ← optimizeMemory(maxKMeans, pKMeans, PO,
       ↪ dataset, OF, HM)
4:     N ← dataset.dimension
5:     rgs ← numeric[N + 1]
6:     for cIt = 0 to NI − 1 do
7:         PAR ← MinPAR + (MaxPAR − MinPAR) ∗ (cIt/NI)
8:         k ← chooseK(maxK, HMCR, PAR, HM)
9:         for i = 0 to N − 1 do
10:            if U(0, 1) ≤ HMCR then
11:                pos ← U(0, HMS − 1)
12:                rgs[i] ← HM[pos][i]
13:                if U(0, 1) ≤ PAR then
14:                    pos ← U(0, N − 1)
15:                    rgs[i] ← HM[0][pos]
16:                end if
17:            else
18:                rgs ← U(0, k − 1)
19:            end if
20:        end for
21:        rgs ← reprocessRGS(rgs)
22:        if U(0, 1) ≤ PO then
23:            rgs ← kmeans(rgs, maxKMeans, pKMeans,
           ↪ distance, dataset)
24:        end if
25:        HM ← replaceTheWorst(HM, rgs, OF)
26:    end for
27:    return HM[0]
28: end function
```

Figura 9: Algoritmo GKR-Registros

En la Figura 10 se muestra el proceso de generación de una nueva armonía con el algoritmo GKR-Registros. En la parte superior se encuentra una matriz que representa una memoria armónica de 5 elementos. En la parte inferior se encuentra una armonía de 10 elementos, donde el último corresponde al valor de fitness. El primer elemento (el de la izquierda) se toma de un elemento del mejor de la memoria, el segundo se toma del segundo elemento de la segunda armonía de la memoria, el tercero, cuarto, quinto y noveno se seleccionan al azar; los elementos sexto, séptimo y octavo se toman de las posiciones respectivas de las

armonías tres, cinco y cuatro en este orden. La nueva armonía no cumple con la restricción de crecimiento, así que después de generada debe ser reprocesada.

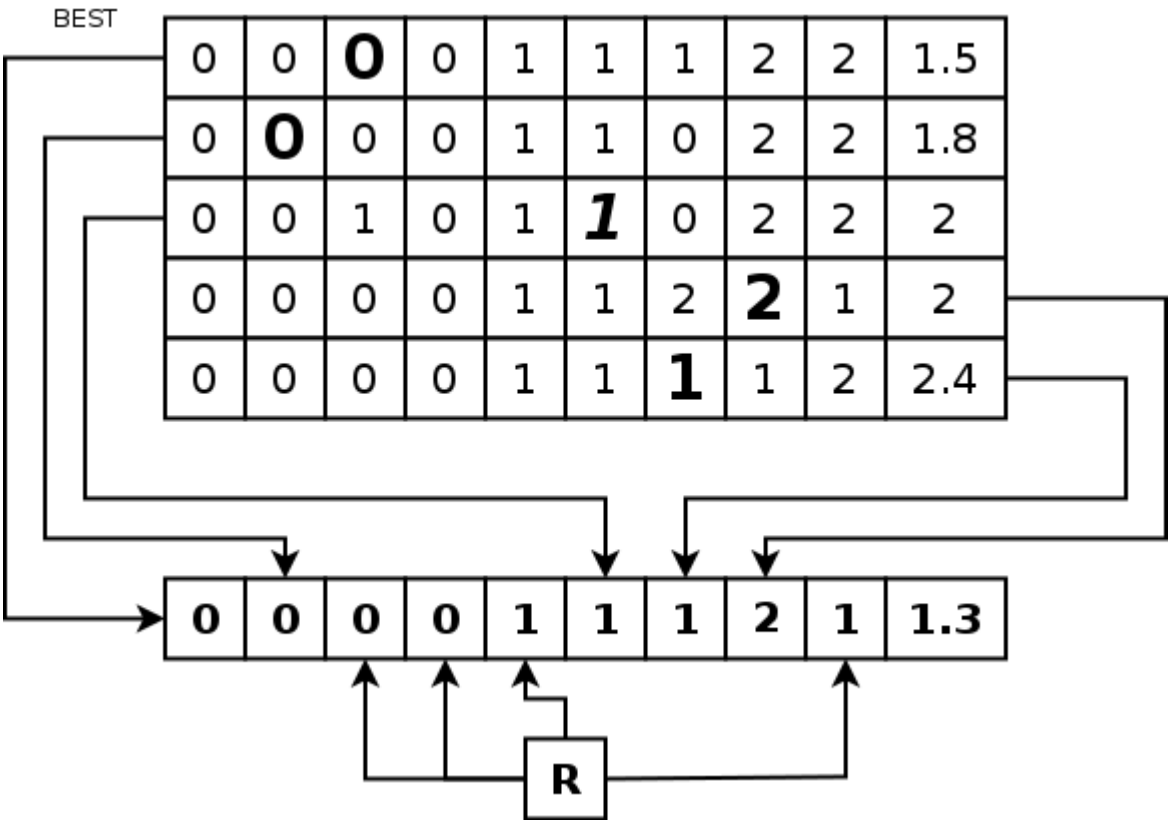

Figura 10: Generación de una armonía con GKR-Registros

El reprocesamiento de la RGS cuyo pseudocódigo se muestra en la Figura 14 consiste en reasignar los elementos del vector de tal manera que cumpla con la restricción de crecimiento. Para ello en primer lugar se recupera el valor mayor de la rgs que se asume es el número de clusters existentes en la rgs, lo cual se hace entre las líneas 2 y 8 en donde a K se le asigna el mayor valor, luego en la línea 9 se incrementa en una unidad. Decimos que se asume que es el número de clusters puesto que durante el proceso de optimización local se pueden perder grupos (unidades muertas) y esto no se ve reflejado en la rgs, por ejemplo, si una rgs con 5 grupos etiquetados del 0 al 4 pierde el tercer y cuarto grupo, los elementos de la rgs tomarán los valores 0, 1 o 4, en una rgs bien formada el mayor valor más 1 indicaría el número de grupos, pero en este caso el valor sería erróneo. Luego se construye una matriz de K filas por dos columnas la cual se utiliza para intercambiar los valores de la rgs actuales por los correctos. Por ejemplo, se tiene una rgs de diez elementos y tres grupos como se muestra en la Figura 11, la matriz de intercambio contendría los valores mostrados en la Figura 12 indicando que si la rgs en la posición i tiene un dos éste se debe reemplazar por el uno. Tras el reprocesamiento se obtiene la rgs mostrada en la Figura 13.

1	2	1	1	2	0	1	1	2	0

Figura 11: RGS sin reprocesar

0	2
1	0
2	1

Figura 12: Matriz de intercambios

0	1	0	0	1	2	0	0	1	2

Figura 13: RGS reprocesada

De manera más detallada, una vez definida la matriz de intercambios se llena de tal manera que los elementos de la primera columna se fijan con su respectivo índice y los elementos de la segunda columna con -1 lo cual se hace en el ciclo a partir de la línea 11. Se recupera en valor de la rgs en la primera posición almacenándolo en una variable temporal tmp y en la posición $[tmp, 1]$ de la matriz de intercambios, así como en la primera posición de la rgs se fija el valor de cero (líneas 15 a 17), esto indica que cuando en la rgs se encuentre el valor tmp se debe reemplazar por cero (En el ejemplo al encontrar un uno en la rgs se debe reemplazar por cero). Se crea una variable max que contiene el mayor valor asignado a la rgs hasta el momento (inicialmente cero) y se fija el valor de K en cero (líneas 18 y 19). Ahora se procede a recorrer la rgs a partir del segundo elemento (es preciso recordar que el primer elemento siempre es cero) y cada vez que se encuentre un valor diferente de -1 en la posición $[i, 1]$ de la matríz se procederá a hacer el intercambio (líneas 21 y 22), si el valor en dicha posición es -1, indicando que no se ha realizado la asignación en la matriz, entonces se incrementa el valor de max y en la posición $[rgs[i], 1]$ de la matriz tanto como en la posición i de la rgs se fija el valor de max (líneas 24 a 26). Utilizando la rgs de la Figura 11 y cuando i vale 1, max incrementaría su valor en 1 y en la posición $[2, 1]$ de la matriz se fijaría el 1, así cada vez que en la rgs se encuentre un dos este valor será reemplazado por uno. Aplicando el proceso recién descrito a la rgs (no válida) de la Figura 11 se obtiene la rgs válida de la Figura 13 la cual puede observarse cumple con la restricción de crecimiento.

Algoritmo reprocessRGS

```
1: function reprocessRGS(rgs)
2:     K ← rgs[0]
3:     N ← rgs.length
4:     for i = 1 to N − 2 do
5:         if K < rgs[i] then
6:             K ← rgs[i]
7:         end if
8:     end for
9:     K + +
10:    swap ← int[K][2]
11:    for i = 0 to K do
12:        swap[i][0] ← i
13:        swap[i][1] ← −1;
14:    end for
15:    tmp ← rgs[0]
16:    swap[tmp][1] ← 0
17:    rgs[0] ← 0
18:    max ← 0
19:    K ← 0
20:    for i = 1 to N − 2 do
21:        if swap[rgs[i]][1] ≠ −1 then
22:            rgs[i] ← swap[rgs[i]][1]
23:        else
24:            max + +
25:            swap[rgs[i]][1] ← max
26:            rgs[i] ← max
27:        end if
28:    end for
29:    return rgs
30: end function
```

Figura 14: Algoritmo reprocessRGS

5.5 ALGORITMO GKR-GRUPOS

En la Figura 15 se presenta el pseudocódigo del algoritmo GKR-Grupos.

Algoritmo GKRGroups

```
1: function GKRGroups(HMS, HMCR, NI, minPAR, maxPAR,
       ↪ PO, maxKMeans, pKMeans, distance,
       ↪ dataset, OF, maxK)
2:     HM ← generateHarmonyMemory(HMS, OF, maxK,
       ↪ dataset, distance)
3:     HM ← optimizeMemory(maxKMeans, pKMeans, PO,
       ↪ dataset, OF, HM)
4:     N ← dataset.dimension
5:     rgs ← numeric[N + 1]
6:     for cIt = 0 to NI − 1 do
7:         PAR ← MinPAR + (MaxPAR − MinPAR) ∗ (cIt/NI)
8:         k ← chooseK(maxK, HMCR, PAR, HM)
9:         for i = 0 to k − 1 do
10:            if U(0, 1) ≤ HMCR then
11:                pos ← U(0, HMS − 1)
12:                nc ← countGroups(hm[pos])
13:                cg ← U(0, nc − 1)
14:                rgs ← mix(rgs, hm[pos], cg)
15:                if U(0, 1) ≤ PAR then
16:                    nc ← countGroups(hm[0])
17:                    cg ← U(0, nc − 1)
18:                    rgs ← mix(rgs, hm[0], cg)
19:                end if
20:            else
21:                tmpRGS ← randRGS(k)
22:                rgs ← mix(rgs, tmpRGS, i)
23:            end if
24:        end for
25:        rgs ← completeRGS(rgs)
26:        rgs ← reprocessRGS(rgs)
27:        if U(0, 1) ≤ PO then
28:            rgs ← kmeans(rgs, maxKMeans, pKMeans,
           ↪ distance, dataset)
29:        end if
30:        HM ← replaceTheWorst(HM, rgs, OF)
31:    end for
32:    return HM[0]
33: end function
```

Figura 15: Algoritmo GKR-Grupos

Este algoritmo se diferencia de GKR-Registros en que las nuevas armonías se crean combinando grupos completos de armonías de la memoria o de una rgs generada de manera aleatoria de acuerdo a las reglas de GBHS, es decir, la nueva armonía no se crea modificando un registro a la vez sino en grupos de registros que pertenecen al mismo clusters (aquellos cuyo valor en la rgs es el mismo). Las diferencias se pueden ver en la línea 9 en la que se indica que el ciclo solo se repetirá de acuerdo al número de grupos seleccionado (valor de K), entre las líneas 11 y 18 en donde se selecciona un armónico de la memoria del cual se escoge un grupo generando un número aleatorio entre cero y el número de grupos (el cual se obtiene con la función countGroups en la línea 16), éste último es combinado con la nueva armonía (líneas 14 o 18 dependiendo del valor de par); y en las líneas 21 y 22 donde se genera una rgs de manera aleatoria la cual se combina con el nuevo armónico indicando con i el cual es el grupo que se desea combinar. Una vez generada una nueva armonía, es posible que algunos de los campos de la rgs hayan quedado vacíos (con un valor de -1), por lo cual se invoca a la función completeRGS (línea 25) en donde se recorre la rgs y al encontrar un valor de -1 se reemplaza por k. En la Figura 16 se muestra el pseudocódigo de algoritmo utilizado para mezclar las rgs que recibe como parámetro dos rgs (rgs1 y rgs2) y un índice de grupo, el cual recorre las dos rgs y si el elemento i de la rgs1 es igual al índice de grupo (línea 3) entonces procede a hacer la combinación, la cual consiste en verificar si el elemento i de la rgs2 está vació (línea 4) en cuyo caso lo fija con el valor del índice de grupo (línea 5), si no está vacío entonces se genera un número entero aleatorio en el intervalo $[0,1]$ (línea 6) y si es uno entonces su valor se reemplaza por el del índice de grupo, caso contrario se deja como estaba.

Algoritmo mix

```
1:  function MIX(RGS1, RGS2, group)
2:      for i = 0 to RGS1.length − 2 do
3:          if RGS1[i] == group then
4:              if RGS2[i] == −1 then
5:                  RGS2[i] ← group
6:              else if randomInt[0, 1] == 1 then
7:                  RGS2[i] = group
8:              end if
9:          end if
10:     end for
11:     return RGS2
12: end function
```

Figura 16: Algoritmo Mix

En la Figura 17 se muestra el proceso de generación de una nueva armonía con el algoritmo GKR-Grupos. En la parte superior se encuentra una matriz que representa una memoria armónica de 5 elementos. En la parte inferior se

encuentran dos estados de una nueva armonía de 10 elementos, donde el último corresponde al valor de fitness. En el paso 1 se selecciona el grupo etiquetado con cero de la segunda armonía de la memoria armónica, como inicialmente la memoria está vacía (todos sus elementos están en -1), simplemente se fijan los elementos del 1 al 4 con los del grupo seleccionado. En el paso 2 se selecciona el grupo etiquetado con uno de la cuarta armonía de la memoria armónica, en este caso las posiciones 5, 6, y 9 se encuentran vacías, así que estos elementos se fijan con los elementos del grupo uno de la memoria seleccionada. En el paso 3 se selecciona el grupo etiquetado con uno de la tercera armonía de la memoria armónica, en este caso las posiciones 3 y 5 ya se encuentran ocupadas, así que se utiliza la función mix para resolver si en las posiciones 3 y 5 se fijan los valores de la armónica seleccionada o se dejan los que están; en este caso la posición 3 se deja como estaba y la posición 5 se resuelve con la de la armonía, aunque en este caso ambas son uno; la séptima posición al estar vacía toma el valor de la memoria seleccionada. Finalmente la posición 8 queda vacía, así que su valor se resuelve con el valor de K seleccionado, que en este caso es de 3.

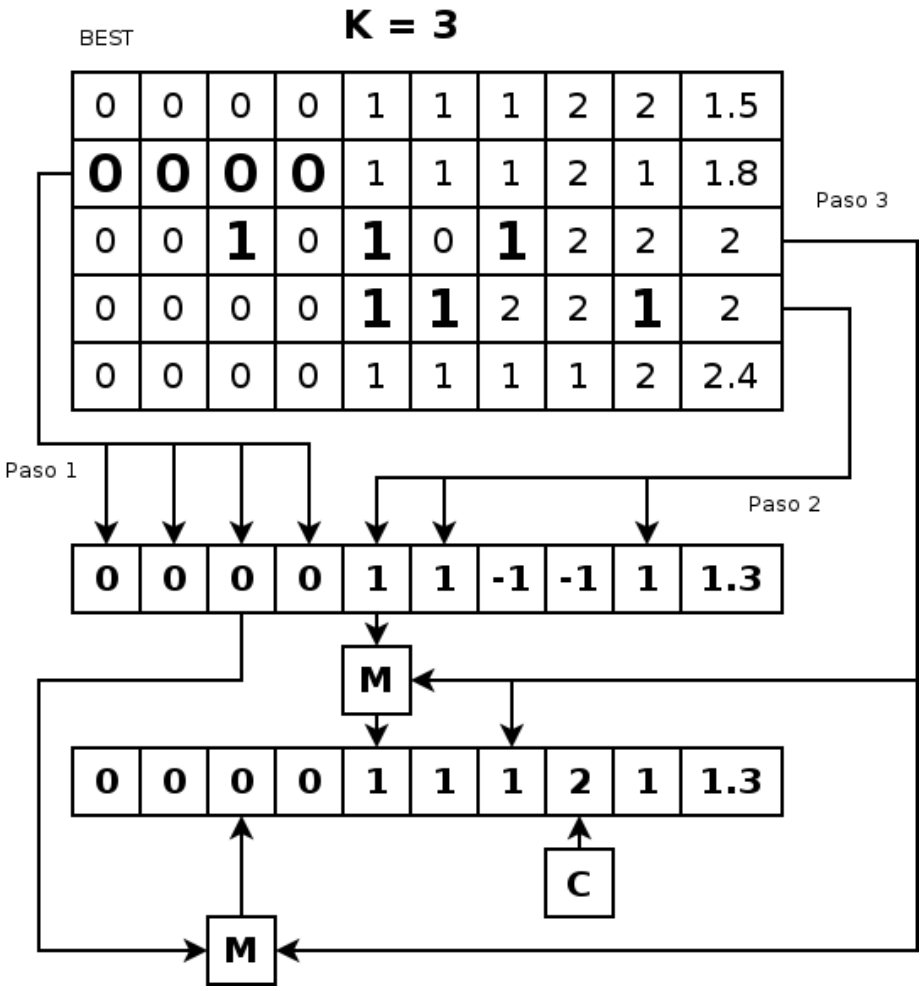

Figura 17: Generación de una armonía con GKR-Grupos

5.6 ALGORITMO GKR-CENTROIDES

En la Figura 19 se presenta el pseudocódigo del algoritmo GKR-Centroides. En este algoritmo el proceso de improvisación se realiza generando un vector de centroides obtenidos de algún elemento de la memoria armónica o de manera aleatoria de acuerdo a las reglas vistas del algoritmo GBHS. Tras la obtención del vector de centroides se genera una rgs asignando cada uno de los elementos del dataset al centroide más cercano (como se hace en una iteración de k-means) y esta es optimizada con k-means si un valor aleatorio generado es menor que PO. Los centroides se calculan promediando los registros de un mismo grupo (líneas 15,19 y 25) y consisten en vectores de números reales de la misma dimensión que los registros del dataset más 1 elemento en el que se almacena el valor de fitness. La función getCentroid recibe como parámetros el armónico y el índice de cluster (valor entre cero y k-1) e internamente recupera todos los elementos del dataset cuyo valor en la rgs coincida con el índice de cluster, calcula el promedio y retorna

el centroide. Se puede ver que el proceso de improvisación (el ciclo a partir de la línea 10) consta de los mismos pasos del GKR base, pero su resultado no es una rgs como tal sino una matriz de centroides así que finalizado el proceso de improvisación se debe asignar cada punto del dataset al centroide más cercano (línea 28), así como se hace en una iteración de k-means, dando como resultado una rgs (la función assign debe llamar a reprocessRGS).

Por ejemplo, si se tiene un dataset con 4 atributos y si la función chooseK retorna 3, al final del proceso de improvisación se obtendrá una matriz de 3 filas por 4 columnas como se muestra en la Figura 18.

0.1	0.22	0.4	0.8
0.9	0.5	0.1	0.4
0.56	0.44	0.16	0.3

Figura 18: Matriz de centroides

Suponiendo que en la primera iteración el centroide se debe recuperar del primer elemento de la memoria armónica (condiciones dadas en las líneas 11 y 16) y que éste tiene 5 grupos (cuyo valor es recuperado en la línea 17 con la función countGroups), en la línea 18 se generaría un número aleatorio cg en el rango [0,4] y en la línea 19 se recuperaría el cg-ésimo centroide del elemento seleccionado, así se obtiene el centroide ubicado en la primera posición de la matriz. Dado que k = 3, el ciclo se repite otras dos veces y finalmente se obtiene la matriz de la Figura 18. Luego se invoca a la función assing de la línea 19, la cual recorre uno a uno los elementos del dataset, calcula la distancia a cada uno de los 3 centroides encontrados, recupera el índice cuya distancia al elemento determinado es menor y lo fija en la posición respectiva en la rgs.

Algoritmo GKRCentroids

```
1: function GKRCentroids(HMS, HMCR, NI, minPAR, maxPAR,
       ↪ PO, maxKMeans, pKMeans, distance,
       ↪ dataset, OF, maxK)
2:     HM ← generateHarmonyMemory(HMSOF, maxK,
       ↪ dataset, distance)
3:     HM ← optimizeMemory(maxKMeans, pKMeans, PO,
       ↪ dataset, OF, HM)
4:     N ← dataset.dimension
5:     rgs ← numeric[N + 1]
6:     for cIt = 0 to NI − 1 do
7:         PAR ← MinPAR + (MaxPAR − MinPAR) ∗ (cIt/NI)
8:         k ← chooseK(maxK, HMCR, PAR, HM)
9:         centroids : double[][]
10:        for i = 0 to k − 1 do
11:            if U(0, 1) ≤ HMCR then
12:                pos ← U(0, HMS − 1)
13:                nc ← countGroups(hm[pos])
14:                cg ← U(0, nc − 1)
15:                centroids[i] ← getCentroid(hm[pos], cg)
16:                if U(0, 1) ≤ PAR then
17:                    nc ← countGroups(hm[0])
18:                    cg ← U(0, nc − 1)
19:                    centroids[i] ← getCentroid(hm[0], cg)
20:                end if
21:            else
22:                tmpRGS ← randRGS(k)
23:                nc ← countGroups(tmpRGS)
24:                cg ← U(0, nc − 1)
25:                centroids[i] ← getCentroid(tmpRGS, cg)
26:            end if
27:        end for
28:        rgs ← assign(centroids, dataset)
29:        if U(0, 1) ≤ PO then
30:            rgs ← kmeans(rgs, maxKMeans, pKMeans)
31:        end if
32:        HM ← replaceTheWorst(HM, rgs)
33:    end for
34:    return HM[0]
35: end function
```

Figura 19: Algoritmo GKR-Centroides

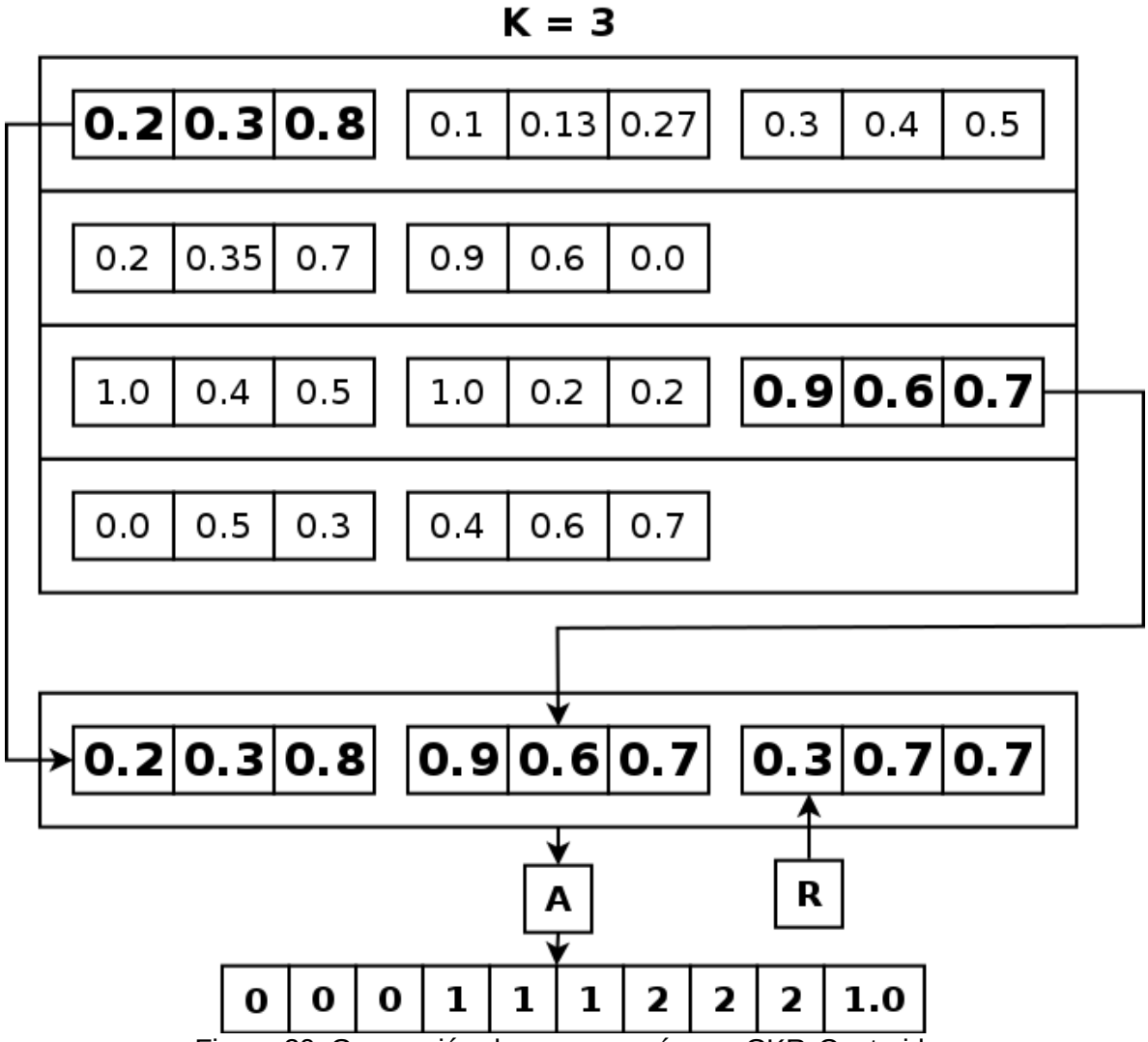

Figura 20: Generación de una armonía con GKR-Centroides

En la Figura 20 se muestra el proceso de generación de una nueva armonía con el algoritmo GKR-Centroides. En la parte superior se encuentra una matriz que representa una memoria armónica de 4 elementos. En la parte inferior se encuentran dos elementos, el primero de ellos es una armonía compuesta por centroides (como las de la memoria armónica), la ultima es una armonía representada por una RGS. De la primera armonía se selecciona el primer centroide y este se fija en la nueva, el segundo centroide es seleccionado de la tercera armonía y se escoge el tercero y el último de los centroides es generado de manera aleatoria. Tras obtener una solución representada por centroides, se asigna cada punto al centroide más cercano, con lo que se obtiene la armonía representada por una RGS.

CAPITULO 6

6. EXPERIMENTACIÓN

En esta sección se presentan los dataset utilizados para la evaluación de los algoritmos propuestos y los algoritmos del estado del arte utilizados para comparar su desempeño. Se muestra el proceso de afinamiento de los parámetros, los resultados obtenidos tanto por los algoritmos propuestos como por los del estado del arte y, finalmente, se realiza un análisis de los resultados obtenidos.

6.1 DESCRIPCIÓN DE LOS DATASETS

A continuación se presenta para cada uno de los datasets el número de elementos y su distribución en cada uno de los grupos; y la etiqueta (o clase) que tiene cada grupo.

6.1.1 DATASET IRIS

Este dataset está compuesto por 150 elementos distribuidos en 3 grupos como se muestra en la Figura 21.

Clase	Iris-setosa	Iris-versicolor	Iris-virginica
Elementos	50	50	50

Figura 21: Asignación de elementos dataset Iris

6.1.2 DATASET GLASS

Este dataset está compuesto por 214 elementos distribuidos en 6 grupos como se muestra en la Figura 22.

Clase	build wind float	build wind non-float	vehic wind float	containers	tableware	headlamps
Elementos	70	76	17	13	9	29

Figura 22: Asignación de elementos dataset Glass

6.1.3 DATASET SONAR

Este dataset está compuesto por 208 elementos distribuidos en 2 grupos como se muestra en la Figura 23.

Clase	Rock	Mine
Elementos	97	111

Figura 23: Asignación de elementos dataset Sonar

6.1.4 DATASET WDBC

Este dataset está compuesto por 569 elementos distribuidos en 2 grupos como se muestra en la Figura 24.

Clase	M	B
Elementos	212	357

Figura 24: Asignación de elementos dataset WDBC

6.1.5 DATASET WINE

Este dataset está compuesto por 178 elementos distribuidos en 3 grupos como se muestra en la Figura 25.

Clase	1	2	3
Elementos	59	71	48

Figura 25. Asignación de elementos dataset Wine

6.2 ALGORITMOS DEL ESTADO DEL ARTE

Para evaluar el desempeño del algoritmo propuesto por un lado se tienen los algoritmos de clustering automático que no precisan conocer el número de grupos, y por el otro los algoritmos de clustering supervisado que si necesitan el número de grupos; y aunque los algoritmos propuestos son del tipo automático es necesario comparar su desempeño con algoritmos supervisados por la escases de

publicaciones en el área de clustering automático, y además por la falta de recursos proporcionados por las publicaciones consultadas dado que muchas de ellas o no brindan información importante con la cual poder repetir los experimentos o utilizan criterios de evaluación poco dicientes, como por ejemplo la suma del error o el valor alcanzado por las funciones objetivo, los cuales no permiten realizar una verdadera comparación de la calidad de los resultados que obtienen cada uno de los algoritmos.

6.2.1 ALGORITMOS DE CLUSTERING AUTOMÁTICO

Se seleccionaron dos algoritmos de clustering automático: GTSCA [60] que es un algoritmo inspirado en el concepto de transposición génico que se define como "Una secuencia de ADN que puede moverse a diferentes posiciones dentro del genoma de una célula" (Adaptado de [60]) y EFCPSO [61] que es un algoritmo basado en PSO (Particle Swarm optimization) que como se vio en la sección dedicada al marco teórico busca simular el comportamiento de enjambres de partículas que se mueven en el espacio. Estos trabajos se seleccionaron debido a que evalúan los algoritmos utilizando la medida recuerdo, que indica el porcentaje de instancias correctamente clasificadas.

6.2.2 ALGORITMOS DE CLUSTERING SUPERVISADO

Se seleccionaron cuatro algoritmos de clustering supervisado: KFA [40] que utiliza el algoritmo firefly (inspirado en el comportamiento de las luciérnagas), PSO+K-Means [62] que utiliza PSO, FCOAC [63] que utiliza el algoritmo de optimización cuckoo y FCM-PSO-MRS [64] que utiliza fuzzy c-means y PSO. Los tres primeros algoritmos utilizan k-means como mecanismo de búsqueda local y su respectiva metaheurística como mecanismo de búsqueda global, el último de ellos a su vez utiliza fuzzy c-means.

6.3 PROCESO DE AFINACIÓN

El proceso de afinación de los parámetros de los algoritmos consiste en generar de manera aleatoria los valores de minPar, maxPar, hmcr, hms, po, nIt, maxK y maxKMeans; generar las tareas como se vio en la sección 4.4.6 con los otros parámetros definidos en el archivo json y la ejecución de las mismas. Finalmente se calcula el promedio de los errores obtenidos en cada ejecución y se almacena en una archivo csv los valores de los parámetros junto al error promedio. Con el parámetro tuneUpIt se determina cuantas veces se ejecuta el proceso de afinación. En el anexo 2 se muestran los valores de parámetros obtenidos tras 57 ejecuciones del proceso de afinación y en la Tabla 1 el valor promedio de cada parámetro. Estos valores promedio son, finalmente, con los que se realizó la experimentación.

minPar	maxPar	hmcr	hms	po	nIt	maxK	maxKMeans	ER
0,52	0,75	0,50	22,21	0,52	613,32	11,54	55,96	38,65

Tabla 1: Valores afinación

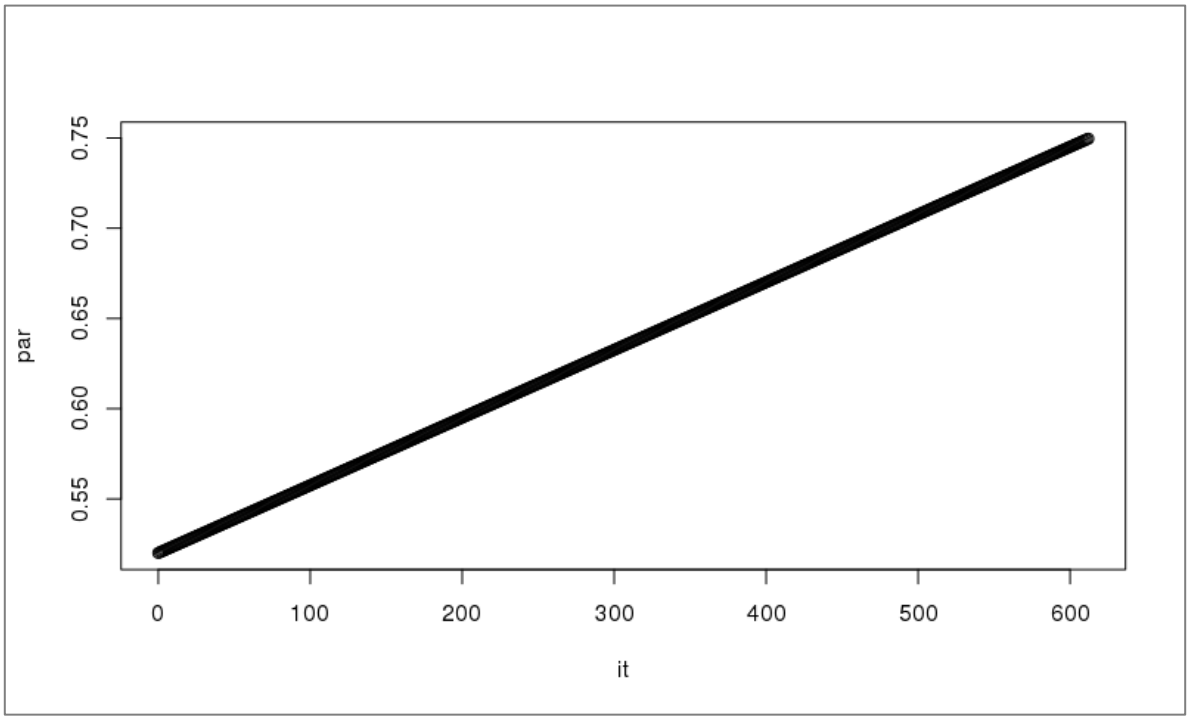

Figura 26: Evolución de PAR

Los valores de la Tabla 1 indican para cada dimensión de una nueva armonía en el proceso de improvisación, que tiene una probabilidad de 0.5 de ser seleccionado de la memoria armónica o de 0.5 de ser generado de manera aleatoria, que la probabilidad condicionada al valor de hmcr de ser seleccionado de una dimensión de la mejor solución es de al menos 0.52 y que esta probabilidad crece de manera lineal conforme avanzan las iteraciones hasta alcanzar un valor máximo de 0.75 como se muestra en la Figura 26; y para una nueva armonía que la probabilidad de ser optimizada con k-means es de 0.52. Los valores de hms, nIt, maxK y maxKMeans indican el tamaño de la memoria armónica, el número de iteraciones de GKR, el valor máximo de grupos y el número máximo de iteraciones de k-means.

6.4 RESULTADOS OBTENIDOS

6.4.1 CLUSTERING AUTOMATICO

En la Tabla 2 se presentan los resultados obtenidos con el algoritmo GKR-Registros, en la Tabla 3 los resultados obtenidos con el algoritmo GKR-Centroides y en la Tabla 4 los resultados obtenidos con el algoritmo GKR-Grupos, todos en modo de clustering automático. En cada tabla se indica por cada fila la función

objetivo y la distancia utilizada y en negrilla el mejor valor obtenido para el dataset de la columna correspondiente. Por ejemplo, para el algoritmo GKR-Registros y el dataset Glass el mejor resultado se obtiene con la función objetivo AIC y la distancia Manhattan (Primera fila y primera columna de la Tabla 2).

Glass	Iris	Sonar	WDBC	Wine	Func	Dist
49.95 ± 3.87	**19.42 ± 10.24**	77.25 ± 1.72	66.02 ± 3.04	**14.47 ± 7.89**	AIC	Manhattan
52.32 ± 3.54	25.33 ± 9.91	51.0 ± 5.24	**13.33 ± 6.96**	26.79 ± 16.46	CHI	Manhattan
55.79 ± 3.41	28.4 ± 9.31	**48.87 ± 6.12**	13.61 ± 11.47	37.09 ± 17.91	AIC	Euclidiana
54.98 ± 1.88	27.48 ± 11.17	52.06 ± 9.37	18.43 ± 14.29	30.52 ± 13.83	CHI	Euclidiana

Tabla 2: Resultados de Error usando GKR-Registros, clustering automático

Glass	Iris	Sonar	WDBC	Wine	Func	Dist
49.36 ± 0.22	**12.66 ± 1.77E-15**	74.98 ± 3.77	61.93 ± 2.84	**17.09 ± 1.53**	AIC	Manhattan
55.59 ± 1.02	33.33 ± 1.42E-14	48.63 ± 0.52	**7.17 ± 0.60**	39.88 ± 1.42E-14	CHI	Manhattan
55.60 ± 1.42E-14	33.33 ± 1.42E-14	44.31 ± 0.17	7.20 ± 3.55E-15	39.88 ± 1.42E-14	AIC	Euclidiana
55.60 ± 1.42E-14	33.33 ± 1.42E-14	**44.23 ± 3.55E-14**	7.20 ± 3.55E-15	39.88 ± 1.42E-14	CHI	Euclidiana

Tabla 3: Resultados de Error usando GKR-Centroides, clustering automático

Glass	Iris	Sonar	WDBC	Wine	Func	Dist
49.06 ± 0.31	**12.66 ± 1.77E-15**	74.50 ± 3.28	61.03 ± 1.67	**17.47 ± 1.85**	AIC	Manhattan
55.23 ± 0.50	33.33 ± 1.42E-14	48.78 ± 0.32	8.16 ± 0.84	39.88 ± 1.42E-14	CHI	Manhattan
55.60 ± 1.42E-14	33.33 ± 1.42E-14	**44.23 ± 3.55E-14**	**7.20 ± 3.55E-15**	39.88 ± 1.42E-14	AIC	Euclidiana

55.60 $\pm$ 1.42E-14	33.33 $\pm$ 1.42E-14	**44.23 $\pm$ 3.55E-14**	**7.20 $\pm$ 3.55E-15**	39.88 $\pm$ 1.42E-14	CHI	Euclidiana

Tabla 4: Resultados de Error usando GKR-Grupos, clustering automático

En la Tabla 5 se muestran los resultados reportados para los algoritmos de clustering automático GTSCA y EFCPSO.

	Glass	Iris	Sonar	WDBC	Wine
GTCSA	62.17 ± 0	13.4 ± 0	-	-	61.1 ± 0
EFCPSO	-	9.33	-	-	22.47

Tabla 5: Resultados GTCSA y EFCPSO

6.4.2 CLUSTERING SUPERVISADO

En la Tabla 6 se presentan los resultados obtenidos con el algoritmo GKR-Registros, en la Tabla 7 los resultados obtenidos con el algoritmo GKR-Centroides y en la Tabla 8 los resultados obtenidos con el algoritmo GKR-Grupos, todos en modo de clustering supervisado. En cada tabla se indica por cada fila la función objetivo y la distancia utilizada y en negrilla el mejor valor obtenido para el dataset de la columna correspondiente.

Glass	Iris	Sonar	WDBC	Wine	Func	Dist
55.95 $\pm$ 0.20	12.66 $\pm$ 1.77E-15	45.19 ± 0	7.73 $\pm$ 5.32E-15	**3.93 $\pm$ 1.33E-15**	AIC	Manhattan
56.01 $\pm$ 0.19	12 ± 0	48.99 $\pm$ 0.14	7.73 $\pm$ 5.32E-15	**3.93 $\pm$ 1.33E-15**	CHI	Manhattan
56.21 $\pm$ 0.21	**11.33 $\pm$ 0**	**43.75 $\pm$ 0**	**7.20 $\pm$ 3.55E-15**	4.81 ± 0.27	AIC	Euclidiana
56.27 $\pm$ 0.50	**11.33 $\pm$ 0**	44.26 $\pm$ 0.11	**7.20 $\pm$ 3.55E-15**	4.49 ± 0	CHI	Euclidiana

Tabla 6: Resultado de error usando GKR-Registros, clustering supervisado

Glass	Iris	Sonar	WDBC	Wine	Func	Dist
49.29 $\pm$ 0.23	12.66 $\pm$ 1.77E-15	45.22 $\pm$ 0.11	**6.76 $\pm$ 0.17**	**3.93 $\pm$ 1.33E-15**	AIC	Manhattan
55.07 $\pm$ 1.8	33.33 $\pm$ 1.42E-14	48.50 $\pm$ 0.73	7.15 ± 0.52	39.88 $\pm$ 1.42E-14	CHI	Manhattan

49.95 $\pm$ 2.45	**11.48 $\pm$ 0.33**	**43.76 $\pm$ 0.08**	6.94 $\pm$ 0.19	4.64 $\pm$ 0.24	AIC	Euclidiana
55.43 $\pm$ 0.51	33.33 $\pm$ 1.42E-14	44.23 $\pm$ 0.99	7.20 $\pm$ 3.55E-15	39.88 $\pm$ 0.71	CHI	Euclidiana

Tabla 7: Resultado de error usando GKR-Centroides, clustering supervisado

Glass	Iris	Sonar	WDBC	Wine	Func	Dist
48.31 $\pm$ 0.95	12.66 $\pm$ 1.77E-15	45.19 $\pm$ 0	7.73 $\pm$ 0.03	**3.93 $\pm$ 1.33E-15**	AIC	Manhattan
48.87 $\pm$ 2.01	33.33 $\pm$ 1.42E-14	48.76 $\pm$ 0.32	7.90 $\pm$ 3.55E-15	39.88 $\pm$ 1.42E-14	CHI	Manhattan
48.03 $\pm$ 2.76	**11.33 $\pm$ 0**	**43.87 $\pm$ 0.21**	**7.20 $\pm$ 3.55E-15**	4.45 $\pm$ 0.24	AIC	Euclidiana
54.31 $\pm$ 1.42	33.33 $\pm$ 1.42E-14	44.23 $\pm$ 3.55E-14	**7.20 $\pm$ 3.55E-15**	39.88 $\pm$ 1.42E-14	CHI	Euclidiana

Tabla 8: Resultado de error usando GKR-Centroides, clustering supervisado

En la Tabla 9 se muestran los resultados reportados para los algoritmos de clustering supervisado KFA, PSO+K-Means, FCOAC y FCM-PSO-MRS.

	Glass	Iris	Sonar	WDBC	Wine
KFA	45.54 $\pm$ 3.37	7.33 $\pm$ 1.10	35.57 $\pm$ 5.55	12.42 $\pm$ 1.02	28.15 $\pm$ 0.95
PSO+M-Means	-	11.8	-	-	13.27
FCOAC	33.35	9.81	-	-	6.18
FCM-PSO-MRS	18.95	10.88	-	-	26.13

Tabla 9: Resultados KFA, PSO+KMeans, FCOA y FCM-PSO-MRS

6.5 ANÁLISIS DE LOS RESULTADOS OBTENIDOS

6.5.1 CLUSTERING AUTOMATICO

Puede verse en las tablas 3 a 5 que las tres versiones del algoritmo propuesto ofrecen soluciones muy similares. GKR-Centroides y GKR-Grupos tienen los resultados más cercanos tanto a nivel de error como de desviación estándar, la diferencia más grande de error se encuentra en el dataset WDBC con CHI como función objetivo y manhattan como función de distancia, cuyo valor es de 0.99 con un valor de 0.23 de diferencia en las desviaciones estándar. El algoritmo GKR-Registros aunque ofrece resultados cercanos a los entregados por los otros dos

algoritmos, puede verse que los valores de error y desviación estándar son mucho más altos. Relacionando el resultado entregado por GKR-Registros para el dataset Wine con CHI como función objetivo y manhattan como función de distancia; con el resultado entregado por GKR-grupos, el valor de error es de 13.08 y de 16.46 para la desviación estándar.

Los resultados entregados por los algoritmos GKR-Centroides y GKR-Grupos son muy cercanos es debido al mecanismo de generación de las nuevas armonías, en el caso de GKR-centroides consiste en traspaso de centroides, que son puntos que representan a todo grupo y en el caso de GKR-grupos consiste en el traspaso de varios elementos pertenecientes a un mismo grupo. Al compartir un centroide entre armonías, con él se comparte la relación existente entre los elementos del mismo grupo; y aunque esta relación se ve afectada por los centroides existentes en la nueva, en el traspaso los elementos estrechamente similares tenderán a permanecer en el mismo grupo. Algo similar ocurre al compartir grupos entre armonías. En el caso del algoritmo GKR-Registros una nueva armonía se genera realizando las asignaciones de las dimensiones una a la vez; en este caso al compartir una dimensión entre una armonía existente y la nueva no se transfiere información respecto a la relación que esta tiene en la armonía original con el resto de elementos aledaños.

Miremos el efecto de la variable PAR. Es necesario enfatizar que la exploración y explotación del espacio de búsqueda está determinada por el valor del parámetro HMCR: si un número aleatorio es menor o igual que HMCR entonces se explota el espacio de búsqueda, es decir, la nueva armonía es modificada teniendo en cuenta los elementos de la memoria armónica, compuesta por las mejores soluciones encontradas en un determinado momento; en caso contrario la nueva armonía es modificada con elementos aleatorios del espacio de búsqueda. Ahora, cuando la armonía es modificada con elementos de la memoria el que lo sea con uno cualquiera o con el mejor depende del valor de PAR. Si un número aleatorio es menor o igual que PAR, la nueva armonía es modificada por valores presentes en la mejor solución. En todas las iteraciones la probabilidad de explotar el espacio de búsqueda es constante, no así el de modificar las nuevas armonías con el contenido de la mejor solución, ya que este valor de Par es dinámico y crece desde minPar hasta maxPar de manera lineal y de acuerdo al valor de la iteración actual. Se puede decir entonces que el valor de par permite que el algoritmo converja a una solución, que no es necesariamente la mejor posible pero si es una solución óptima, y gracias a HMCR sin evitar dejar de explorar el espacio de búsqueda. Los valores similares de error en los tres algoritmos nos indican que todos tienden a converger a una solución particular en el espacio de búsqueda, pero que GKR-Registros y GKR-Grupos explotan mejor el espacio de búsqueda que GRK-Registros.

Vemos en las tablas 3 a 5 el efecto de la función objetivo y la función de distancia sobre el resultado del proceso de clustering. En la Tabla 4 en el dataset Glass la diferencia entre resultados es pequeña para los valores extremos, siendo de 6.23

para el error y 0.79 para la desviación estándar; pero para el resto de los datasets estas diferencias son considerables. Para el dataset Iris la diferencia es de 20.66, para el dataset Sonar es de 30, para el dataset WDBC es de 54 y para el dataset Wine es de 22.79. Vemos además que de los 15 resultados con el valor de error más bajo (17 si se tiene en cuenta los resultados repetidos para Sonar y WDBC obtenidos con GKR-grupos con AIC y CHI como funciones objetivo y distancia euclidiana), 9 se obtienen con distancia Manhattan, de los cuales a su vez 6 se obtienen con la función objetivo CHI. Aunque CHI y Manhattan son una buena combinación, no hay que pasar por alto el hecho de que para los datasets Sonar y WDBC los peores resultados se obtienen precisamente con CHI y Manhattan. Las funciones objetivos aquí utilizadas, que son las que determinan el número de grupos, no tienen forma de trabajar con las diferentes formas que pueden tomar los grupos y la forma en la que trabaja K-means, que es minimizando la suma del error, hace que los grupos formados alrededor de un centroide sean hiperesféricos. Por esta razón si los datasets no son hiperesféricos o sino están bien delimitados o poco solapados, los resultados no pueden ser los mejores.

Respecto al desempeño de los algoritmos desarrollados en comparación a los de clustering automático seleccionados de la literatura, vemos que ofrecen mejores resultados con Wine y Glass que GTCSA; y que EFCPSO con Wine, a excepción de GKR-Registros cuyo valor de error es superior con el dataset Iris, pero que ninguno mejora el resultado reportado para EFCPSO con Iris.

6.5.2 CLUSTERING SUPERVISADO

Vemos en este caso que los resultados obtenidos con los tres algoritmos mejoran bastante respecto al clustering automático. Por ejemplo en el algoritmo GKR-centroides con el dataset WDBC, la función objetivo AIC y la distancia de Manhattan el error pasa de 61.93 ± 2.8 a 6.76 ± 0.17, es decir, el error disminuye en 55.17 puntos. Lo mismo ocurre con wine con la misma configuración del anterior, pasando de 17.09 ± 1.53 a $3.93 \pm 1.33\text{E-}5$, que representa una disminución en 13.16 puntos.

Respecto a los resultados obtenidos con GKR-Centroides y GKR-grupos, puede verse que los resultados siguen siendo similares. La mayor diferencia entre resultados se encuentra con el dataset Glass, la distancia de manhattan y la función objetivo CHI, con un valor de 5.48 ± 0.3, el resto de diferencias están por debajo de 1.9. Como se vió en la sección anterior, estos resultados se dan por la similitud que tienen estos dos algoritmos.

Ahora bien, aunque conocer el número de grupos constituye una gran ventaja en el proceso de clústering, la calidad de los resultados está supeditada a la función objetivo, la función de similitud entre elementos (en este caso función de distancia) y a las figuras de los grupos. Por ejemplo, al trabajar con el dataset Iris y CHI si con esta función objetivo se llega al número correcto de grupos, que es tres, el error es de 11.33, pero si se llega a encontrar una solución en el proceso de

improvisación que tenga dos grupos, el error llega a 33.33 aunque el valor de fitness para esa armonía es menor que para cualquiera con tres grupos y aunque los algoritmos en este caso trabajan con un número de K fijo e igual al valor real, durante el proceso de optimización con k-means se pierden grupos, por eso se puede llegar a tener menos grupos que los configurados. Como se vio en la sección anterior, las funciones objetivo y las distancias seleccionadas no están diseñadas para trabajar con formas no esféricas.

Los algoritmos desarrollados en este proyecto trabajando en modo supervisado son altamente competitivos con los del estado del arte, cosa que se evidencia en las diferencias entre resultados. KFA presenta mejores resultados para los datasets Glass, Iris y Sonar, pero los resultados entregados por los algoritmos propuestos son mejores para los datasets WDBC y Wine. PSO+KMeans supera los resultados en Iris, pero no con Wine. FCOAC y FCM-PSO-MRS presentan mejores resultados en Glass e Iris, pero resultados inferiores con Wine.

CAPITULO 7

7. CONCLUSIONES Y TRABAJOS A FUTURO

7.1 CONCLUSIONES

En este proyecto de investigación se implementó un algoritmo que combina Global-best Harmony Search como mecanismo de optimización global, K-Means de optimización local y RGS de representación de las soluciones. A partir de ahí se exploraron otras dos alternativas que modifican el proceso de improvisación de una nueva armonía permitiendo la asignación masiva de elementos a grupos con lo que se mantiene las relaciones existentes entre elementos. Tales asignaciones se hacen en el algoritmo original, llamado GKR-Registros, de una dimensión (o posición) de la armonía a la vez, en los algoritmos GKR-Centroides y GKR-Grupos se hacen por centroides y grupos de elementos respectivamente.

Los algoritmos desarrollados son afectados por la forma de los grupos; por un lado porque K-Means solo genera grupos hiperesféricos debido a la función de distancia base (Minkowski), por otro lado a que las funciones objetivo están basadas en la compactibilidad y separación, como es el caso de CHI y en la suma del error cuadrado como es el caso de AIC, que es calculada como la distancia entre los puntos y su centroide más cercano. Como se pudo apreciar en los diagramas de dispersión, los datasets utilizados presentan formas no esféricas, no están bien compactos o están muy solapados.

El uso de una metodología orientada al desarrollo proyectos de investigación permite orientar las actividades desarrolladas a la solución de problemas de manera sistemática, dedicando recursos al conocimiento en profundidad de los problemas encontrados, a la búsqueda de soluciones fundadas en conocimiento generado en proyectos desarrollados por terceros en campos relacionados y a la validación de los resultados obtenidos de manera objetiva. Esto permite que los resultados del proyecto sean repetibles y que con ello puedan ser validados por otros investigadores interesados en el estudio de temas relacionados y que se evite la ejecución de tareas innecesarias cuyos resultados, o no aportan al proyecto, o lo desvían hacia otros caminos no relacionados. El que la metodología sea iterativa permite dividir el proyecto en partes más pequeñas y más fáciles de manejar; y el que sea incremental permite planificar el proyecto de tal manera que en cada iteración se obtienen resultados evaluables y cada vez más cercanos al propósito final.

El trabajo aquí presentado es un buen punto de partida para el desarrollo de otros algoritmos de clustering particional de datos. El uso de GBHS para la búsqueda global mostró ser un buen punto de partida para el algoritmo K-Means, el cual

depende de la inicialización para entregar buenos resultados; y RGS es un buen mecanismo de representación y de limitación del espacio de búsqueda.

De acuerdo a los resultados obtenidos y recordando la pregunta de investigación que dice lo siguiente:

¿Es posible obtener un algoritmo para clustering de datos basado en Global-best Harmony Search, K-means como optimizador local, Restricted Growth Strings y criterios de calidad internos como AIC, que sea competitivo en calidad con los reportados por otros algoritmos del estado del arte, pero con menor costo computacional y sin definir previamente el número de grupos (clustering automático)?

Se concluye que sí es posible encontrar un algoritmo para clustering de datos basado en Global-best Harmony Search, utilizando K-Means como optimizador local y Restricted Growth Strings como mecanismo de representación de las soluciones utilizando criterios de calidad como AIC. Su desempeño es altamente competitivo con el desempeño de varios algoritmos del estado del arte y estos resultados se obtienen sin definir de antemano el número de grupos como se vio en la sección dedicada al análisis de resultados. Respecto al costo computacional, con el algoritmo K-Means no se tiene manera de limitar el espacio de búsqueda a soluciones no visitadas ni a partes en las que sea más factible el encontrar buenas soluciones, por lo que se debe hacer una ejecución exhaustiva para asegurar la consecución de buenas soluciones, lo cual no es necesario hacer en este caso debido a que con la metaheurística y la representación de las soluciones se aseguran buenos puntos de partida para K-Means.

7.2 TRABAJOS FUTUROS

Entre los trabajos que pueden desarrollarse se encuentran aquellos orientados a la mejora de los resultados y aquellos orientados a la mejora del rendimiento. Aunque es posible trabajar en ambos aspectos, se recomienda abordar cada uno a la vez debido a la complejidad asociada.

Los trabajos orientados a la mejora de resultados están relacionados a la búsqueda de alternativas que permitan trabajar con las diferentes formas que pueden tomar los grupos. En este caso se puede experimentar con otras funciones objetivo, con otras funciones de similitud, con variaciones al algoritmo, con la implementación de mecanismos para la evaluación multiobjetivo o a la integración de otros algoritmos de búsqueda local.

Por otra parte, aquellos enfocados en la mejora del rendimiento deben tratar con datasets de alta dimensionalidad y gran número de elementos. Los datasets utilizados se puede decir que son de complejidad media, tanto por la forma de los grupos como por la dimensionalidad, especialmente de glass que es tamaño 60 y la de wine que es de tamaño 30. Así que un trabajo a futuro es el abordar el clustering de big data desde el enfoque aquí propuesto. Esto implicaría la

adopción de estrategias de optimización de los cálculos redundantes como los cálculos de distancias entre elementos o la adopción de estrategias para memoria cache.

CAPITULO 8

[1] K. A. A. Nazeer y M. P. Sebastian, “Improving the Accuracy and Efficiency of the k-means Clustering Algorithm”, *World Congr. Eng. 2009 Vol. 1*, pp. 308–312, ene. 2009.

[2] S. Prasanna y E. Maran, “Stock Market Prediction Using Clustering with Meta-Heuristic Approaches”, *Gazi Univ. J. Sci.*, vol. 28, núm. 3, pp. 395–403, jul. 2015.

[3] Z. Zhang, J. Zhang, y H. Xue, “Improved K-Means Clustering Algorithm”, en *Image and Signal Processing, 2008. CISP ’08. Congress on*, 2008, vol. 5, pp. 169–172.

[4] T. E. Colanzi *et al.*, “Application of Bio-inspired Metaheuristics in the Data Clustering Problem”, *CLEI Electron. J.*, vol. 14, núm. 3, pp. 6–6, dic. 2011.

[5] C. C. Aggarwal y C. K. Reddy, *Data Clustering: Algorithms and Applications*. Chapman \& Hall/CRC, 2013.

[6] X. Wu *et al.*, “Top 10 algorithms in data mining”, *Knowl. Inf. Syst.*, vol. 14, núm. 1, pp. 1–37, 2007.

[7] J. Kogan, C. Nicholas, y M. Teboulle, *Grouping Multidimensional Data: Recent Advances in Clustering*. Springer-Verlag New York, Inc., 2006.

[8] S. J. Nanda y G. Panda, “A survey on nature inspired metaheuristic algorithms for partitional clustering”, *Swarm Evol. Comput.*, vol. 16, pp. 1–18, jun. 2014.

[9] D. Arthur y S. Vassilvitskii, “k-means++: The advantages of careful seeding”, en *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007, pp. 1027–1035.

[10] C. Cobos, M. Mendoza, y E. León, “A harmony search algorithm for clustering with feature selection”, *Rev. Fac. Ing. Univ. Antioquia*, pp. 153–164, 2010.

[11] Tajunisha y Saravanan, “An efficient method to improve the clustering performance for high dimensional data by Principal Component Analysis and modified K-means”, *Int. J. Database Manag. Syst. IJDMS*, vol. 3, 2011.

[12] M. A. Rahman, M. Z. Islam, y T. Bossomaier, “ModEx and Seed-Detective: Two novel techniques for high quality clustering by using good initial seeds in K-Means”, *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 27, núm. 2, pp. 113–128, 2015.

[13] M. G. H. Omran y M. Mahdavi, “Global-best harmony search”, *Appl. Math. Comput.*, vol. 198, núm. 2, pp. 643–656, 2008.

[14] S. S. Skiena, *The Algorithm Design Manual*. London: Springer London, 2008.

[15] H. D. Acquah, “On the Comparison of Akaike Information Criterion and Consistent Akaike Information Criterion in Selection of an Asymmetric Price Relationship: Bootstrap Simulation Results”, *Agris -Line Pap. Econ. Inform.*, vol. 5, núm. 1, pp. 3–9, mar. 2013.

[16] “Information on Set Partitions”. [En línea]. Disponible en: http://theory.cs.uvic.ca/inf/setp/SetPartitions.html. [Consultado: 19-oct-2015].

[17] "Binomial Coefficient -- from Wolfram MathWorld". [En línea]. Disponible en: http://mathworld.wolfram.com/BinomialCoefficient.html. [Consultado: 19-oct-2015].
[18] M. Gendreau y J.-Y. Potvin, Eds., *Handbook of Metaheuristics*, vol. 146. Boston, MA: Springer US, 2010.
[19] Zong Woo Geem, Joong Hoon Kim, y G. V. Loganathan, "A New Heuristic Optimization Algorithm: Harmony Search", *SIMULATION*, vol. 76, núm. 2, pp. 60–68, 2001.
[20] K. S. Lee y Z. W. Geem, "A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice", *Comput. Methods Appl. Mech. Eng.*, vol. 194, núm. 36–38, pp. 3902–3933, sep. 2005.
[21] M. Mahdavi, M. Fesanghary, y E. Damangir, "An improved harmony search algorithm for solving optimization problems", *Appl. Math. Comput.*, vol. 188, núm. 2, pp. 1567–1579, may 2007.
[22] D. Zou, L. Gao, J. Wu, S. Li, y Y. Li, "A novel global harmony search algorithm for reliability problems", *Sched. Healthc. Ind. Syst.*, vol. 58, núm. 2, pp. 307–316, mar. 2010.
[23] J. Kennedy y R. Eberhart, "Particle swarm optimization", en *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 1995, vol. 4, pp. 1942–1948 vol.4.
[24] M. McCool, A. D. Robison, y J. Reinders, "Chapter 11 - K-Means Clustering", en *Structured Parallel Programming*, M. McCool, A. D. Robison, y J. Reinders, Eds. Boston: Morgan Kaufmann, 2012, pp. 279–289.
[25] B. D. (University of P. O.-L. Modal'X), *clusterCrit: Clustering Indices*. 2016.
[26] P. Sánchez Orellana, J. Torres Jiménez, y C. Castellanos Sánchez, "Restricted Growth String for Video-Type Classification", en *Advances in Computational Intelligence*, vol. 116, W. Yu y E. Sanchez, Eds. Springer Berlin Heidelberg, 2009, pp. 519–526.
[27] M. M. Öztürk, U. Cavusoglu, y A. Zengin, "A novel defect prediction method for web pages using k-means++", *Expert Syst. Appl.*, vol. 42, núm. 19, pp. 6496–6506, nov. 2015.
[28] V. Tunali, T. Bilgin, y A. Camurcu, "An Improved Clustering Algorithm for Text Mining: Multi-Cluster Spherical K-Means.", *Int. Arab J. Inf. Technol. IAJIT*, vol. 13, núm. 1, pp. 12–19, ene. 2016.
[29] K. Hornik, I. Feinerer, M. Kober, y C. Buchta, "Spherical k-means clustering", *J. Stat. Softw.*, vol. 50, núm. 10, pp. 1–22, 2012.
[30] S. Zhong, "Efficient online spherical k-means clustering", en *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, 2005, vol. 5, pp. 3180–3185 vol. 5.
[31] G. Perim, E. Wandekokem, y F. Varejão, "K-Means Initialization Methods for Improving Clustering by Simulated Annealing", en *Advances in Artificial Intelligence – IBERAMIA 2008*, vol. 5290, H. Geffner, R. Prada, I. Machado Alexandre, y N. David, Eds. Springer Berlin Heidelberg, 2008, pp. 133–142.

[32] Ting Su, “A Deterministic Method for Initializing K-Means Clustering”, en *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, 2004, vol. 0, pp. 784–786.
[33] M. Lichman, *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences, 2013.
[34] T. E. Colanzi, W. K. G. Assuncao, A. T. R. Pozo, A. C. B. K. Vendramin, y D. A. B. Pereira, “Empirical Studies on Application of Genetic Algorithms and Ant Colony Optimization for Data Clustering”, *Chil. Comput. Sci. Soc. SCCC 2010 XXIX Int. Conf. Of*, pp. 1–10, nov. 2010.
[35] U. Aickelin, D. Dasgupta, y F. Gu, “Artificial Immune Systems”, en *Search Methodologies*, E. K. Burke y G. Kendall, Eds. Springer US, 2014, pp. 187–211.
[36] P. Merz y B. Freisleben, “A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem”, en *Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99*, 1999, vol. 3, p. 2070–Vol. 3.
[37] “R: Wilcoxon Rank Sum and Signed Rank Tests”. [En línea]. Disponible en: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/wilcox.test.html. [Consultado: 31-oct-2015].
[38] Z. Zainuddin, K. H. Lai, y P. Ong, “Wavelet neural networks initialization using hybridized clustering and harmony search algorithm: Application in epileptic seizure detection.”, *AIP Conf. Proc.*, vol. 1522, núm. 1, pp. 251–260, abr. 2013.
[39] A. Ahmadyfard y H. Modares, “Combining PSO and k-means to enhance data clustering”, en *International Symposium on Telecommunications, 2008. IST 2008*, 2008, pp. 688–691.
[40] T. Hassanzadeh y M. R. Meybodi, “A new hybrid approach for data clustering using firefly algorithm and K-means”, en *2012 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP)*, 2012, pp. 007-011.
[41] X.-S. Yang, “A New Metaheuristic Bat-Inspired Algorithm”, en *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, J. R. González, D. A. Pelta, C. Cruz, G. Terrazas, y N. Krasnogor, Eds. Springer Berlin Heidelberg, 2010, pp. 65–74.
[42] J. Ji, W. Pang, Y. Zheng, Z. Wang, y Z. Ma, “A Novel Artificial Bee Colony Based Clustering Algorithm for Categorical Data.”, *PLoS ONE*, vol. 10, núm. 5, pp. 1–17, may 2015.
[43] Z. Huang, “Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values”, *Data Min Knowl Discov*, vol. 2, núm. 3, pp. 283–304, 1998.
[44] D. Karaboga, “An idea based on honey bee swarm for numerical optimization”, Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, 2005.
[45] Y. Yang, “An evaluation of statistical approaches to text categorization”, *Inf. Retr.*, vol. 1, núm. 1–2, pp. 69–90, 1999.

[46] W. M. Rand, “Objective criteria for the evaluation of clustering methods”, *J. Am. Stat. Assoc.*, vol. 66, núm. 336, pp. 846–850, 1971.
[47] A. José-García y W. Gómez-Flores, “Automatic clustering using nature-inspired metaheuristics: A survey”, *Appl. Soft Comput.*, vol. 41, pp. 192–213, abr. 2016.
[48] K. S. Pratt, “Design Patterns for Research Methods: Iterative Field Research”, en *AAAI Spring Symposium: Experimental Design for Real-World Systems*, 2009.
[49] A. Jain, K. Nandakumar, y A. Ross, “Score normalization in multimodal biometric systems”, *Pattern Recognit.*, vol. 38, núm. 12, pp. 2270–2285, dic. 2005.
[50] M. Orlov, “Efficient generation of set partitions”, *Eng. Comput. Sci. Univ. Ulm Tech Rep*, 2002.
[51] “A000110 - OEIS”. [En línea]. Disponible en: https://oeis.org/A000110. [Consultado: 03-nov-2016].
[52] D. Pelleg y A. W. Moore, “X-means: Extending K-means with Efficient Estimation of the Number of Clusters”, en *Proceedings of the Seventeenth International Conference on Machine Learning*, San Francisco, CA, USA, 2000, pp. 727–734.
[53] “Weka 3 - Data Mining with Open Source Machine Learning Software in Java”. [En línea]. Disponible en: http://www.cs.waikato.ac.nz/~ml/weka/. [Consultado: 16-dic-2016].
[54] “bobhancock/goxmeans”, *GitHub*. [En línea]. Disponible en: https://github.com/bobhancock/goxmeans. [Consultado: 16-dic-2016].
[55] C. Cobos, J. Andrade, W. Constain, M. Mendoza, y E. León, “Web document clustering based on Global-Best Harmony Search, K-means, Frequent Term Sets and Bayesian Information Criterion”, en *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.
[56] “JUnit - About”. [En línea]. Disponible en: http://junit.org/junit4/. [Consultado: 23-ene-2017].
[57] “R: The R Project for Statistical Computing”. [En línea]. Disponible en: https://www.r-project.org/. [Consultado: 23-ene-2017].
[58] “¿Qué es NoSQL (No Solo SQL)? - Definición en WhatIs.com”, *SearchDataCenter en Español*. [En línea]. Disponible en: http://searchdatacenter.techtarget.com/es/definicion/NoSQL-No-Solo-SQL. [Consultado: 09-feb-2017].
[59] “Data Science Platform”, *RapidMiner*. [En línea]. Disponible en: https://rapidminer.com/. [Consultado: 02-may-2017].
[60] R. Liu, L. Jiao, X. Zhang, y Y. Li, “Gene transposon based clone selection algorithm for automatic clustering”, *Inf. Sci.*, vol. 204, pp. 1–22, oct. 2012.
[61] Y. Zheng, Y. Zhou, y J. Qu, “An Improved PSO Clustering Algorithm with Entropy-Based Fuzzy Clustering”, *WSEAS Trans. Comput.*, vol. 14, pp. 88–89, 2015.
[62] G. Saini y H. Kaur, “A Novel Approach Towards K-Mean Clustering Algorithm With PSO”, *Int. J. Comput. Sci. Inf. Technol.*, vol. 5, 2014.

[63] E. Amiri y S. Mahmoudi, “Efficient protocol for data clustering by fuzzy Cuckoo Optimization Algorithm”, *Appl. Soft Comput.*, vol. 41, pp. 15–21, abr. 2016.

[64] Y. Xianfeng y L. Pengfei, “Tailoring Fuzzy C-Means Clustering Algorithm for Big Data Using Random Sampling and Particle Swarm Optimization”, *Int. J. Database Theory Appl.*, vol. 8, núm. 3, pp. 191–202, jun. 2015.

[65] Y. Liu, Z. Li, H. Xiong, X. Gao, J. Wu, y S. Wu, “Understanding and Enhancement of Internal Clustering Validation Measures”, *IEEE Trans. Cybern.*, vol. 43, núm. 3, pp. 982–994, jun. 2013.