E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Estudio comparativo de modelos genéticos en la tercera fase de un clasificador FARC-HD

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor: Sisniega Soriano Iñigo

Director: Delgado Sanz José Antonio

Pamplona, 5/12/2022

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

RESUMEN

En el presente proyecto se han explorado, implementado y comparado diferentes modelos de algoritmos genéticos además de sus correspondientes operadores genéticos y parámetros asociados como alternativas al algoritmo CHC que opera en la tercera fase de un clasificador FARC-HD (Fuzzy Association Rule-Base Classification Model for High-Dimensional problems) efectuando un proceso genético de post-procesamiento de selección de reglas y ajuste lateral. Uno de los principales objetivos de este proyecto es encontrar un modelo que sea capaz de encontrar un buen equilibrio entre la exploración y explotación sobre las regiones de soluciones y evitar en la medida de lo posible una convergencia prematura en mínimos locales, así como disponer de un buen rendimiento en espacios de búsqueda complejos.

PALABRAS CLAVE

- Lógica difusa

- Sistema de clasificación basados en reglas difusas

- FARC-HD

- Computación evolutiva

- Inteligencia de enjambre

- Algoritmos genéticos

- Evolución diferencial

- PSO

ÍNDICE

1 INTRODUCCIÓN

La tecnología de la información y la comunicación ha crecido de manera exponencial en las últimas dos décadas y como consecuencia, hoy en día somos capaces de recopilar cantidades sustanciales y complejas de datos.

Según Hilbert y López solo en el periodo de 1986 hasta 2007 la información almacenada a escala mundial creció a un ritmo anual de un 23% [1]. Esta "revolución de la información" supone uno de los mayores retos en la comunidad científica, un cambio de pensamiento que se aleje del enfoque tradicional que poseíamos hasta el momento para el tratamiento y análisis de datos de modo que las técnicas implementadas para estudiar el big data sean escalables en el tiempo y memoria además de en la medida de lo posible computacionalmente poco costosas.

Uno de los objetos de estudio derivados de este crecimiento exponencial en el área de aprendizaje supervisado es el problema de clasificación de datos en espacios de alta dimensión o High-Dimensional spaces, caracterizado por tener conjuntos de datos con miles de variables que hacen en muchos casos inviable el uso de una gran parte de las técnicas de clasificación existentes.

Es por esto por lo que en la literatura van surgiendo con el paso del tiempo nuevas técnicas de clasificación y mejoras en los sistemas ya existentes mucho más robustas que son capaces de operar sobre grandes conjuntos de datos con un óptimo uso de recursos.

Este es el caso de los sistemas de clasificación basados en reglas difusas [2] en los que nos vamos a centrar en este proyecto, ampliamente utilizados por su gran interpretabilidad. En concreto nos vamos a centrar en una de sus alternativas para conjuntos complejos de datos, los sistemas de clasificación basados en reglas de asociación difusas para espacios de alta dimensión con selección genética de reglas y ajuste lateral (FARC-HD) propuesto en [3] que pretende integrar en un solo modelo el descubrimiento de asociaciones ocultas en los datos y la minería de reglas de clasificación que permitan predecir la clase para nuevos conjuntos de datos.

La propuesta que realizamos en este proyecto es una búsqueda y estudio de varias alternativas al algoritmo evolutivo CHC implementado en la fase de selección de reglas y ajuste lateral del clasificador FARC-HD, capaces de mantener o incluso mejorar el rendimiento del clasificador sobre una serie de conjuntos de datos de alta dimensionalidad.

1.1 OBJETIVOS

El objetivo principal de este proyecto es encontrar un algoritmo evolutivo alternativa a CHC (que es el que implementa por defecto el clasificador FARC-HD) capaz de encontrar un buen equilibrio entre la exploración y explotación sobre las regiones de soluciones y evitar en la medida de lo posible una convergencia en mínimos locales, así como disponer de un buen rendimiento en espacios de búsqueda de alta dimensionalidad y complejidad.

Para alcanzar nuestro objetivo se hace imprescindible fragmentar el proyecto en diferentes tareas, podemos establecer estas tareas intermedias como las siguientes:

- La primera tarea será realizar un estudio previo mediante varias lecturas teóricas que nos familiarizarán con conceptos sobre lógica difusa y reglas de asociación, también nos aportarán conocimientos sobre el funcionamiento de los clasificadores basados en reglas difusas, así como del clasificador modificado para operar con espacios de alta dimensionalidad FARC-HD, por último, se realizarán varias lecturas sobre los algoritmos que vamos a implementar (Algoritmo genético clásico, Diferencial Evolution y Particle Swarm Optimization).

- En la segunda tarea se hará un estudio del código que implementa el clasificador FARC-HD junto con CHC en Java que será la base sobre la que realizaremos las modificaciones de este proyecto y sobre la que realizaremos la experimentación y evaluación de los resultados.

- La tercera tarea consistirá en implementar los algoritmos escogidos para sustituir a CHC mencionados en la anterior tarea por medio del entorno de desarrollo integrado NetBeans [4].

- En la cuarta tarea desarrollaremos un script en el lenguaje de programación Pyhton utilizando el entorno informático interactivo Jupyter Notebook [5] que nos permitirá realizar la experimentación de este proyecto. Este script realizará una ejecución de los modelos genéticos implementados alterando sus hiperparámetros sobre un conjunto de datasets de manera que nos permita observar sus comportamientos en diferentes escenarios y evaluar qué modelo y selección de hiperparámetros son la mejor alternativa.

1.2 ORGANIZACIÓN

La memoria de este proyecto se organizará en tres partes fundamentales:

1. **Preliminares:** En esta parte se expondrá todo el marco teórico en el que se fundamenta el proyecto, así como todas las técnicas utilizadas para su desarrollarlo.

2. **Experimentación y resultados:** Es en esta parte donde tendrá lugar la explicación del marco experimental de este proyecto, la descripción de los datasets escogidos y de las decisiones tomadas, la presentación de los resultados de la experimentación y su análisis. Además, se indicará la mejor solución encontrada y se discutirá sobre su desempeño en este tipo de sistemas y si resulta una alternativa realista.

3. **Conclusión y líneas futuras:** Será el cierre de la memoria en el que se expondrá una breve reflexión del proyecto realizado, las conclusiones a las que se han llegado tras la experimentación y las vías de investigación que surgen del trabajo realizado.

2 PRELIMINARES

En esta sección se va a presentar de manera detallada el marco teórico del proyecto exponiendo con especial detalle los sistemas de clasificación basados en reglas difusas, el modelo FARC-HD y los algoritmos evolutivos implementados junto con sus operadores genéticos.

2.1 MACHINE LEARNING

El aprendizaje automático o Machine Learning (ML) [6] es el campo de estudio dentro de la inteligencia artificial [7] en el que se desarrollan algoritmos y modelos estadísticos que permiten que los sistemas informáticos aprendan en base a la experiencia y a partir de un conjunto de datos, es decir, dota a los sistemas informáticos la capacidad de realizar una tarea específica sin necesidad de ser programados explícitamente.

Cuando hablamos de aprendizaje automático podemos distinguir tres tipos generales de modelos:

- **Aprendizaje supervisado:** En este tipo de modelos se ajusta una función que realiza predicciones en base a pares entrada-salida partiendo de un conjunto de datos previamente etiquetados.

- **Aprendizaje no supervisado:** En este tipo de modelos no hay un conocimiento a priori, el aprendizaje del modelo se realiza en base a un conjunto de datos de entrada del que se extraen patrones y estructuras de los que a primera vista no se tiene información.

- **Aprendizaje semisupervisado:** Este tipo de modelos combina los modelos supervisados y no supervisados, utiliza datos de entrenamiento tanto etiquetados como datos no etiquetados que son de más fácil obtención y pueden ser muy beneficiosos para el aprendizaje.

En nuestro caso, vamos a aplicar técnicas de aprendizaje supervisado, vamos a partir de conjuntos de datos (datasets) previamente etiquetados y nuestro problema se va a centrar en conseguir un modelo capaz de etiquetar nuevos datos. Este tipo de problema es conocido como **clasificación**.

2.2 CLASIFICACIÓN

La clasificación es un problema dentro del aprendizaje supervisado en el que se parte de un conjunto de ejemplos $e_i \in E$ descritos mediante un conjunto de características o variables y previamente etiquetados en un conjunto de clases o etiquetas conocidas cj $\in C = \{C_1, \dots, C_N\}$ y se obtiene un modelo (conjunto de reglas de decisión, probabilidades o funciones) capaz de predecir la clase de nuevos ejemplos (datos de prueba) con la mayor precisión posible.

Si el conjunto de clases solo toma dos posibles valores, estaremos hablando de una clasificación binaria mientras que si hay múltiples clases posibles estaremos frente a un problema de clasificación multiclase, en este proyecto se trabajarán con conjuntos de datos de ambos tipos.

Además, deberemos tener en cuenta que muchas veces existirá un desbalanceo entre las clases donde tendremos conjuntos de datos en los que las frecuencias relativas de las clases no estarán equilibradas lo que sesgará el clasificador hacia la clase mayoritaria y otras veces las clases no serán del todo separables perjudicando de manera directa en la precisión de nuestro clasificador.

Un factor decisivo a la hora de resolver problemas de clasificación es la elección del algoritmo de clasificación más adecuado para el problema que deseamos resolver, esta elección depende principalmente de la naturaleza del problema que estemos tratando. En la literatura existen un gran número de algoritmos aplicables a la clasificación, algunos de los más clásicos son las máquinas de soporte vectorial (SVM) [8], los árboles de decisión [9], Naive Bayes [10], k-NN [11] o las redes neuronales [12].

En este proyecto, vamos a centrarnos en los sistemas de clasificación basados en reglas difusas o SCBRDs. Este tipo de clasificadores son una fusión entre la teoría de conjuntos difusos y los sistemas basados en reglas clásicos que utilizan reglas IF-THEN en las que el antecedente de la regla está compuesto por sentencias de lógica difusa. Son ampliamente utilizados por su gran interpretabilidad para el usuario final y su alto rendimiento, además son capaces de trabajar con incertidumbre e información incompleta lo que los hace buenos candidatos para su aplicación a problemas reales donde pueden existir ambigüedades e información inconsistente [13].

Pero primero, para entender estos sistemas y qué ventajas proporciona la incorporación de la teoría de conjuntos difusos frente a los sistemas de reglas clásicos es necesaria una breve introducción a conceptos básicos sobre lógica difusa.

2.3 LÓGICA DIFUSA Y TEORÍA DE CONJUNTOS DIFUSOS

La lógica borrosa [14] [15] [16] ha sido ampliamente utilizada en la resolución de problemas de alta complejidad en los que la información habitualmente es imprecisa, vaga y subjetiva. Esto ha propiciado que en los campos de la inteligencia artificial y el reconocimiento de patrones haya sido de gran utilidad, en parte gracias a su capacidad de imitar el modo de toma de decisiones y razonamiento humano y su gran adaptabilidad al mundo real en el que vivimos que es en definitiva intrínsecamente difuso.

La teoría de conjuntos difusos fue desarrollada por Lotfi Asker Zadeh en el año 1965 [17] como una extensión de la lógica de conjuntos clásica donde definía un conjunto borroso como una clase de objetos con un continuo de grados de pertenencia que oscila entre cero y uno y una función de pertenencia que asigna a cada objeto del conjunto un grado de pertenencia determinado. Zadeh basaba su idea en la premisa de que en el mundo real las clases de objetos a menudo son imprecisas y no pertenecen a un conjunto exclusivamente si no que pueden pertenecer a múltiples conjuntos con cierto grado de verdad, sobre todo cuando son clases que expresan ideas ambiguas y subjetivas como por ejemplo “la clase de las personas bellas”.

Cuando hablamos de conjuntos clásicos (o también llamados conjuntos crisp para diferenciarlos de los conjuntos difusos), dado un universo de discurso X y siendo A y B conjuntos de este, podemos definir:

I. **Función de pertenencia** como:

$$\mu_A: X \rightarrow \{0,1\} \mid \mu_A(x) = \begin{cases} 1 \; cuando \; x \; \in \; A \\ 0 \; cuando \; x \; \notin A \end{cases}$$

II. Operaciones básicas como:

-Intersección: $x \in A \cap B$ si $x \in A$ y $x \in B$
-Unión: $x \in A \cup B$ si $x \in A$ o $x \in B$
-Complemento: $x \in \bar{A}$ si $x \notin A$

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

Pero cuando hablamos de conjuntos difusos, la gran diferencia respecto a los conjuntos clásicos reside en que la función de pertenencia se encuentra generalizada de modo que en vez de asignar para cada $x \in X$ un valor dentro del conjunto {0,1}, lo hace dentro del intervalo [0,1]. En la figura 1 podemos observar con un ejemplo sencillo el concepto de conjunto difuso (figura de la derecha) donde no hay fronteras precisas.

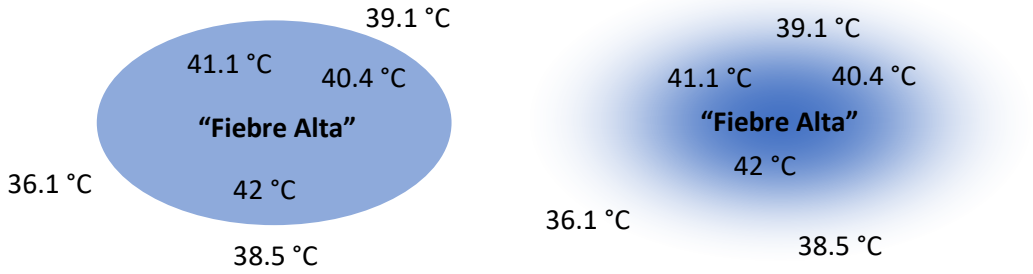

Figura 1: Conjunto crisp y conjunto difuso correspondientes a fiebre alta

En definitiva, podemos definir formalmente un conjunto difuso A en el universo de discurso X como un mapeo:

$$A: X \rightarrow [0,1]$$

$$\mu_A: X \rightarrow [0,1]$$

De manera que cualquier elemento del universo de discurso X pertenece al conjunto difuso A con un cierto grado de pertenencia $\mu_A \in [0,1]$, y A queda completamente determinado por:

$$A = \{(x, \mu_A(x)) \mid x \in X\}$$

A continuación, se describe un ejemplo (figura 2) que expone todos los elementos de la variable lingüística edad cuyo universo de discurso estará formado por los valores comprendidos entre 0 y 100 años y cuyos valores lingüísticos serán "joven" "adulto" y "maduro".

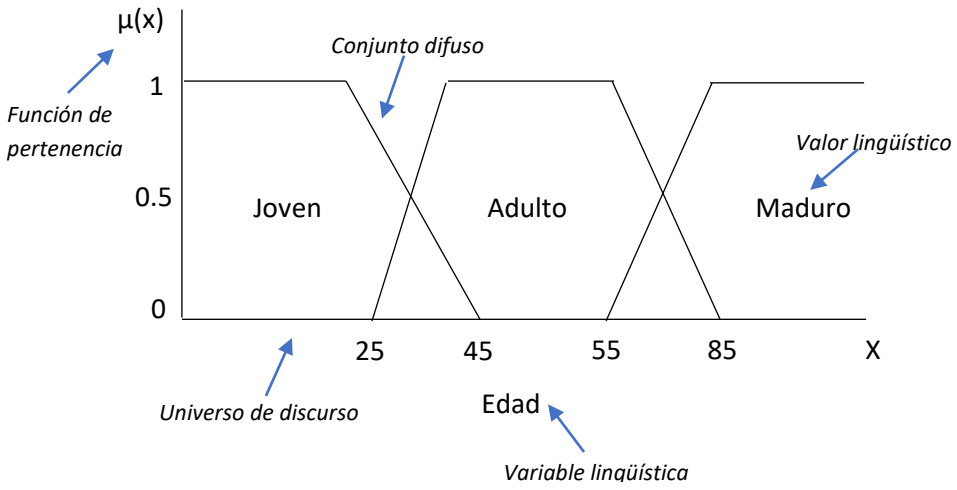

Figura 2: Ejemplo de conjuntos difusos de la variable lingüística edad

Además, podemos observar cómo existe un solapamiento entre conjuntos difusos en los que un elemento posee grados de permanencia en múltiples conjuntos. Esto nos señala que el elemento posee características asociadas a ambos conjuntos.

Pero como ya hemos mencionado previamente, los clasificadores con los que vamos a trabajar en este proyecto operan con reglas de tipo IF-THEN en las que se aplican los conjuntos difusos, por ello, para entender este tipo de reglas y como se realiza su construcción, es necesaria una breve introducción al **razonamiento difuso o aproximado**.

A diferencia con la lógica clásica, en la lógica difusa podemos inferir un consecuente aun que el antecedente de la regla no se satisfaga plenamente si no que las reglas se satisfarán con un cierto grado de cumplimiento del antecedente, esto es lo que denominamos razonamiento aproximado [18].

En la construcción de las reglas, llamamos proposición difusa simple a aquella proposición que asigna un valor lingüístico a una variable lingüística. Continuando con el ejemplo anterior sobre la edad, podemos describir "la edad de Juan es Joven" como una posible preposición difusa simple. Además, podemos formar proposiciones difusas compuestas agrupando dos o más proposiciones difusas simples mediante los operadores lógicos AND y OR y la negación.

Si a esta idea le sumamos la definición del significado de la implicación dentro del marco de conjuntos difusos, seremos capaces de inferir reglas de tipo IF-THEN utilizadas en los sistemas basados en reglas difusas.

Las dos implicaciones más extendidas son la implicación de Mamdani y la implicación de Zadeh. Siendo A y B conjuntos difusos definidos sobre los universos de discurso X e Y respectivamente y funciones de pertenencia $\mu_A(x)$ Y $\mu_B(y)$ se definen:

III. Implicación de Mamdami:

$$\mu_{A\to B}(x,y) = \min\big(\mu_A(x), \mu_B(y)\big) \;\forall\, x \in X, \qquad \forall\, y \in Y$$

IV. Implicación de Zadeh:

$$\mu_{A\to B}(x,y) = \mathrm{s}\big(t\big(\mu_A(x), \mu_B(y)\big), n(\mu_A(x))\big) \;\forall\, x \in X, \qquad \forall\, y \in Y$$

2.4 SISTEMAS DE CLASIFICACIÓN BASADOS EN REGLAS DIFUSAS

Los sistemas de clasificación basados en reglas difusas o SCBRDs [19] [20] [21] son el resultado del matrimonio entre el diseño de los sistemas basados en reglas y la lógica difusa, están compuestos por reglas del tipo IF-THEN en las que el antecedente y el consecuente están formados por proposiciones de lógica difusa.

Este tipo de sistemas se fundamentan en el diseño de los sistemas expertos que simulan el razonamiento humano de la misma manera en la que lo haría un experto en un área de conocimiento determinada. Debido a que adopta su diseño, al igual que en los sistemas expertos, los SCBRDs se componen de dos subsistemas principales: La base de conocimiento y el motor de inferencia.

La **base de conocimiento** es el componente encargado de almacenar toda la información representada mediante observaciones y reglas que son extraídas del problema que queremos resolver, está formada por una base de datos (BD) y una base de reglas (BR). La base de datos contiene las variables lingüísticas del problema, los valores lingüísticos (o etiquetas lingüísticas) utilizados en la construcción de las reglas y los conjuntos difusos asociados a ellos además de las funciones de pertenencia. La base de reglas contiene el conjunto de reglas en forma de sentencias IF-THEN, formadas por las variables y los valores lingüísticos de la base de datos necesarias para la clasificación.

El **motor de inferencia** por su parte se compone de un interfaz de fuzzificación y un sistema de inferencia. El interfaz de fuzzificación se encarga de la transformación de los valores de entrada nítidos o crisp (por lo general valores reales) en valores dentro del marco de conjuntos difusos y el sistema de inferencia es el encargado, a partir de esas entradas difusas transformadas y haciendo uso de la información de la base de conocimiento, de aplicar un **método de razonamiento difuso** (MRD) que infiere una salida (clase).

En la figura 3 se representa el esquema general de un SCBRDs, donde se puede observar el recorrido que efectuaría un nuevo patrón de entrada. El nuevo patrón entra en el sistema a través de la interfaz de fuzzificación ubicada en el motor de inferencia donde se convierte en una entrada difusa, imprescindible para el sistema de inferencia. Posteriormente la entrada es recibida por el sistema de inferencia que gracias a su interacción con la base de conocimiento y su MRD nos retorna una salida.

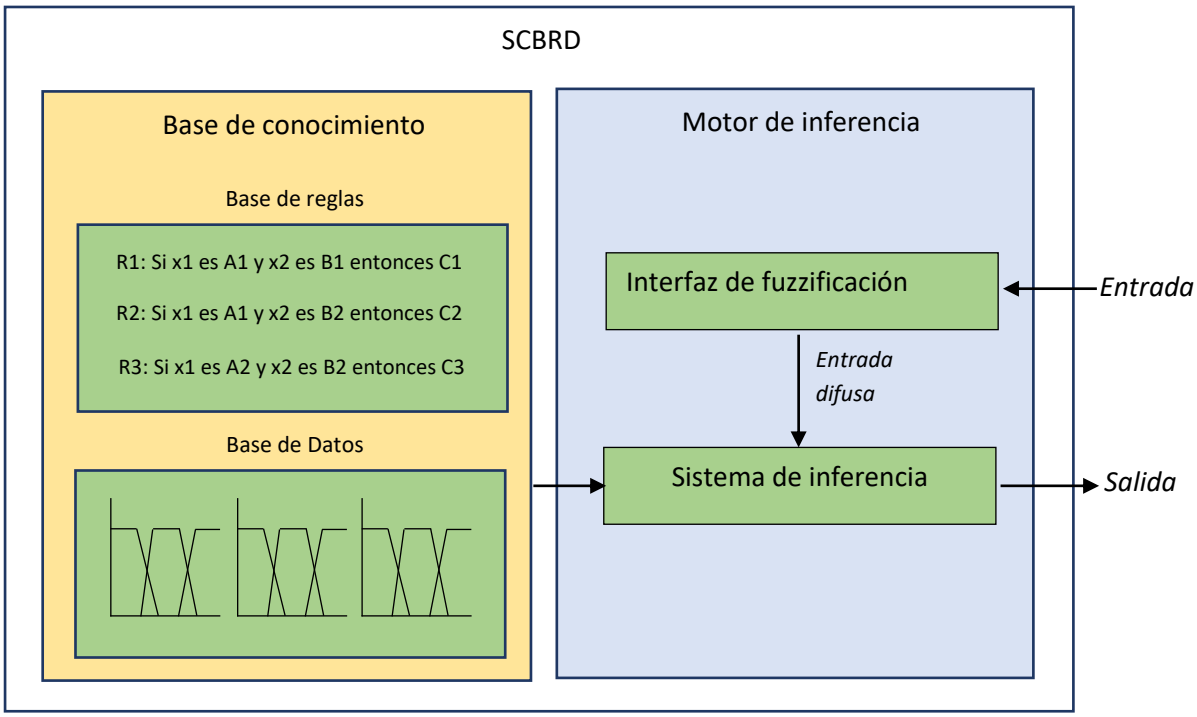

Figura 3: Esquema de un SCBRD

Pero, para entender mejor cómo se infiere una salida a partir de un nuevo patrón, a continuación, se expondrá en detalle el funcionamiento del método de razonamiento difuso.

Un método de razonamiento difuso o MRD es el proceso por el que se infiere una conclusión o resultado a partir de un patrón de entrada y un conjunto de reglas difusas. En los sistemas de clasificación basados en reglas difusas en concreto, el resultado obtenido es una de las posibles clases del problema.

A diferencia de los métodos de razonamiento clásicos en los que es imprescindible la compatibilidad total del patrón de entrada con el antecedente de las reglas para poder inferir un resultado, en los MRD es posible inferir un resultado en base a un grado de compatibilidad, lo que les convierte en una herramienta muy potente para modelar el razonamiento humano. Cabe destacar que la elección del MRD adecuado para nuestro problema es crucial para obtener un buen rendimiento en el clasificador.

Consideremos un problema de clasificación con N variables y un conjunto de M clases $C = \{C_1, \dots, C_M\}$, donde un nuevo patrón del que deseamos inferir su clase es de la forma $P = (x_1, \dots, x_N)$. Además, consideremos una base de reglas compuesta por un conjunto de L reglas difusas $R = \{R_1, \dots, R_L\}$ de uno de los siguientes tipos:

1. **Reglas difusas con una clase en el consecuente:**

$$R_k: Si\ x_1\ es\ A_1^k\ y\ \dots y\ x_N\ es\ A_N^k\ entonces\ Y\ es\ C_j$$

Donde $x_1, \dots, x_N$ son el conjunto de variables lingüísticas que definen un patrón, $A_1^k, \dots, A_N^k$ son los valores lingüísticos utilizados para representar los conjuntos difusos del problema, C_j es la clase a la que pertenece el patrón e Y es la variable lingüística de salida.

2. **Reglas difusas con una clase y un grado de certeza en el consecuente:**

$$R_k: Si\ x_1\ es\ A_1^k\ y\ \dots y\ x_N\ es\ A_N^k\ entonces\ Y\ es\ C_j\ con\ grado\ r^k$$

Donde la regla es prácticamente igual que la anterior, pero con el añadido del grado de certeza en el consecuente asociado a la clase C_j.

3. **Reglas difusas con grados de certeza asociados a cada una de las clases en el consecuente:**

$$R_k: Si\ x_1\ es\ A_1^k\ y\ \dots y\ x_N\ es\ A_N^k\ entonces\ (r_1^k, \dots, r_M^k)$$

Donde r_j^k es el grado de certeza de la regla R_k asociado a la clase C_j.

Nótese que, si en el tercer tipo de reglas tomamos como valores para $(r_1^k, \dots, r_M^k)$ como $r_v^k = 1\ y\ r_j^k = 0\ \ \forall\ j = 1, \dots, M\ siendo\ j \neq v$ tendremos una regla del primer tipo mientras que si tomamos $r_v^k = r_k\ y\ r_j^k = 0\ \ \forall\ j = 1, \dots, M\ siendo\ j \neq v$ tendremos una regla del segundo tipo, es decir, el tercer tipo de reglas podemos considerarlo como una generalización de los otros dos tipos.

Podemos definir un modelo general de MRD para la clasificación por medio de cuatro etapas fundamentales:

I. **Obtención del grado de compatibilidad:** En esta etapa se calcula el grado de compatibilidad del patrón de entrada con el antecedente de todas las reglas de la BR. El grado de compatibilidad nos indica la fuerza de activación del antecedente de todas las reglas con el patrón dado y se calcula aplicando una t-norma a los grados de pertenencia de los elementos del patrón a los conjuntos difusos, es decir, es una agregación de todos los grados de pertenencia de un patrón concreto.

$$R^k(p) = T\left(\mu_{A_1^k}(x_1), \dots, \mu_{A_N^k}(x_N)\right) \qquad \forall\ k = 1, .., L$$

II. **Obtención del grado de asociación:** En esta etapa, haciendo uso del grado de compatibilidad calculado en la etapa anterior y combinándolo con el grado de certeza asociado a cada regla, se calcula el grado de asociación del patrón con cada una de las M clases en función de cada regla de la BR. La función h frecuentemente en la literatura se ha tomado como el operador mínimo o el operador producto.

$$\mathrm{b_j^k} = \mathrm{h}\left(R^k(p), r_j^k\right) \quad \forall\, j = 1, .., M \;\; \forall\, k = 1, \dots, L$$

III. **Aplicación de la función de agregación:** En esta etapa se utiliza una función de agregación que combina los grados de asociación positivos por clases calculados en el paso anterior. El resultado de esta agregación nos indica el grado de fuerza que tiene el patrón dado para clasificarlo en cada una de las distintas clases. Como norma general, la función de agregación suele ser una t-conormas, pero otros operadores han sido utilizados como la media aritmética o la suma normalizada.

$$Y_j = f\left(\mathrm{b_j^k}, \mathrm{b_j^k} > 0\right) \quad \forall\, j = 1, .., M \;\; \forall\, k = 1, \dots, L$$

IV. **Clasificación:** Una vez calculado el grado de asociación o "fuerza" del patrón con cada una de las clases, se aplica una función de decisión F que asigna al patrón la clase que tenga mayor grado de asociación con él. Esta será la salida del método de razonamiento difuso y el resultado de la clasificación que brindará nuestro clasificador basado en reglas difusas.

$$C_i = F(Y_1, \dots, Y_M)\ donde\ Y_i = max_{j=1,..,M}\, Y_j$$

Por último, una vez entendido el diseño y funcionamiento de un SCBRD hay que describir el esquema básico del proceso de aprendizaje que debe realizar uno de estos sistemas para poder llegar a clasificar nuevos patrones. Podemos dividir este proceso en tres tareas principales:

- **Selección de variables (opcional):** Selección de las variables del problema que va a tener en cuenta el SCBRD, si no realizásemos este paso, todas las variables disponibles serían utilizadas.

- **Selección del MRD**: En base al problema que estemos tratando selección del método de razonamiento difuso más adecuado.

- **Obtención de la base de conocimiento:**

    - Generación de las particiones difusas: Llamamos partición difusa al conjunto de valores lingüísticos y conjuntos difusos que definen una variable lingüística. Esta es la información que se almacena en la BD (véase figura 3).

    - Generación de las reglas difusas: Generación de las reglas difusas que van a formar parte de la base de reglas.

    - Selección de un subconjunto de reglas difusas (opcional): Selección del menor conjunto de reglas de la BR posible perdiendo el mínimo rendimiento en la clasificación y conservando la máxima cantidad de información.

    - Ajuste de las particiones difusas (opcional): los conjuntos difusos de una partición difusa pueden ser ajustados lateralmente para mejorar el rendimiento del clasificador.

Uno de los principales problemas de los SCBRDs es el crecimiento exponencial del espacio de reglas difusas que sufren cuando el número de variables del problema que estamos tratando es elevado, este hecho produce problemas de escalabilidad y aumenta de manera muy significativa el coste computacional de este tipo de clasificadores.

Para abordar este gran problema, han surgido en la literatura diversas variantes del SCBRD básico que implementan técnicas de preprocesamiento como la selección de variables y técnicas de postprocesamiento como la reducción de reglas entre otras, que permiten obtener clasificadores de bajo coste computacional y una alta precisión. Este es el caso del clasificador FARC-HD sobre el que vamos a trabajar en este proyecto.

2.5 FARC-HD

FARC-HD o Fuzzy Association Rule-Based Classification for High-Dimensional problems [3] es un modelo de clasificación muy preciso e interpretable basado en los SCBRDs y en el diseño de la clasificación asociativa capaz de operar con espacios de búsqueda complejos.

2.5.1 REGLAS DE ASOCIACIÓN DIFUSAS Y CLASIFICACIÓN

Las reglas de asociación [22] son utilizadas para describir dependencias entre elementos de una misma base de datos que permiten detectar cuando la ocurrencia de uno de ellos está asociada a la ocurrencia de otros elementos en la misma transacción. Son expresiones de la forma $A \rightarrow B$ donde A y B son conjuntos de elementos y $A \cap B = \emptyset$, es decir, si todos los elementos de A ocurren en una transacción, todos los elementos de B también ocurrirán en la transacción.

Podemos definir además las medidas más importantes a la hora de seleccionar una regla de asociación, estas medidas son el soporte y la confianza. Dados A y B dos conjuntos de elementos, podemos definir el soporte y la confianza como:

- **Soporte de una regla de asociación:** Es el porcentaje de transacciones que contienen tanto a A como a B. Se calcula como el número de transacciones que contienen a $A \cup B$ (σ) entre el número de transacciones totales de la base de datos:

$$Supp(A \rightarrow B) = Supp(A \cup B\ ) = \frac{\sigma(A \cup B)}{N}$$

- **Confianza de una regla de asociación:** Es el porcentaje de transacciones que contienen a A en las que también está incluido B, dicho de otra manera, es la frecuencia con la que aparece B en las transacciones en las que aparece A.

$$Conf(A \rightarrow B) = \frac{Supp(A \cup B\ )}{Supp(A)}$$

Si a estas reglas de asociación les agregamos el uso de conjuntos difusos en el antecedente y las adaptamos para problemas de clasificación haciendo que el consecuente de la regla contenga una etiqueta de clase, entonces obtendremos **reglas de asociación difusas para la clasificación** utilizadas en el modelo FARC-HD.

Reescribiendo las medidas para la selección de reglas anteriores y adaptándolas a las reglas de asociación difusas para la clasificación [23] tendremos que las nuevas medidas serán de la forma siguiente:

- **Soporte de una regla de asociación difusa para la clasificación:**

$$Supp(A \rightarrow C_k) = \frac{\sum_{x_p \in Class\ C_k} \mu_A(x_p)}{N}$$

- **Confianza de una regla de asociación difusa para la clasificación:**

$$Conf(A \rightarrow C_k) = \frac{\sum_{x_p \in Class\ C_k} \mu_A(x_p)}{\sum_{p=1}^{N} \mu_A(x_p)}$$

Donde N es el número de patrones etiquetados del problema y $\mu_A(x_p)$ es el grado de compatibilidad del patrón x_p con el antecedente de la regla difusa calculado mediante la operación producto (primer paso del MDR general).

Como resultado, el clasificador FARC-HD operará con reglas de asociación difusas para la clasificación de la forma:

$$Regla\ R_k: Si\ x_1\ es\ A_1^k\ y \ldots y\ x_m\ es\ A_m^k\ entonces\ Clase = C_k\ con\ RW_k$$

Donde R_k es la etiqueta de la regla k-ésima, $P = (x_1, \ldots, x_m)$ es un patrón de entrada representado mediante un vector m-dimensional, A_i^k representa un conjunto difuso antecedente, C_k es la etiqueta de la clase y RW_k o Rule Weight es el peso asociado a la regla calculado mediante el valor de confianza difusa o factor de certeza (CF):

$$RW_k = CF_k = \frac{\sum_{x_p \in Class\ C_k} \mu_{A_k}(x_p)}{\sum_{p=1}^{N} \mu_{A_k}(x_p)}$$

Donde $\mu_{A_k}(x_p)$ es el grado de compatibilidad del patrón x_p con el antecedente de la regla difusa R_k.

Y para la clasificación de nuevos patrones hará uso del MRD del voto ponderado o combinación aditiva en la que cada regla de la base de reglas proporciona un voto para la clase de su consecuente, esta es calculada siguiendo el tercer paso del modelo de MRD general descrito en la sección anterior donde se toma la suma como función de agregación.

2.5.2 PROCESO DE APRENDIZAJE DE UN CLASIFICADOR FARC-HD

El proceso de aprendizaje que permite obtener un clasificador FARC-HD está basado en tres etapas:

I. Extracción de reglas de asociación difusa para la clasificación.
II. Preselección de reglas candidatas.
III. Selección genética de reglas y ajuste lateral.

Etapa 1. Extracción de reglas de asociación difusa para la clasificación:

En esta primera etapa se genera la base de reglas (BR) utilizando un árbol de búsqueda que enumera todos los posibles conjuntos difuso de una clase para posteriormente formar las reglas difusas de clasificación.

Este árbol de búsqueda se construye siguiendo un algoritmo Apriori adaptado para la clasificación, este algoritmo se basa en la propiedad a priori que dicta que, si un conjunto de elementos (itemset) es frecuente también lo serán todos sus subconjuntos, este hecho es debido a la anti-monotonía del soporte, es decir, se puede afirmar que el soporte de un conjunto de elementos nunca puede ser mayor que el de ninguno de sus subconjuntos. Dado un conjunto de elementos X, decimos que es frecuente si su soporte es igual o mayor al soporte mínimo establecido:

$$Supp(X) \geq MinSupp$$

Donde el mínimo soporte para cada clase se define como:

$$MinSuppC_k = minSupp * f_{C_k}$$

Siendo $minSupp$ el mínimo soporte determinado por un experto y f_{C_k} es la proporción de patrones de la clase C_k.

El algoritmo Apriori es aplicado tantas veces como clases tenga el problema que estemos resolviendo. El algoritmo comienza inicializando un árbol de búsqueda donde el nodo raíz o nivel 0 contiene un conjunto vacío. Posteriormente en el primer nivel del árbol se forman los nodos que contendrán a todos los valores lingüísticos posibles para cada variable del problema, en este nivel cada nodo representará un conjunto de elementos de un solo elemento (valor lingüístico de la variable). A continuación, se determinan para cada conjunto de elementos del primer nivel el soporte del conjunto y la confianza de la regla que se generaría, si un nodo no es frecuente o si su confianza es mayor que la confianza mínima establecida MinConf (la regla ha alcanzado un nivel de calidad satisfactorio) el árbol no se expandirá más por ese nodo y por tanto no se generarán subconjuntos de este. Para generar los siguientes niveles el algoritmo

combina un conjunto de elementos de un nodo del nivel anterior con los de los nodos que pueden ampliarse siempre que se puedan combinar. Una vez generado el árbol y obtenidos todos los conjuntos de elementos frecuentes se construyen las reglas de asociación candidatas para la clasificación estableciendo los conjuntos de elementos frecuentes como el antecedente de la regla y la clase correspondiente como consecuente.

Con objeto de generar una base de reglas interpretable para el usuario final, se introduce además una profundidad máxima del árbol (MaxDepth) que limita el número de antecedente de las reglas difusas generadas.

Etapa 2. Preselección de reglas difusas candidatas:

Uno de los inconvenientes del algoritmo A priori es que puede generar una gran cantidad de reglas, por ello, esta segunda etapa realiza una selección de las reglas más importantes mediante el uso de un esquema de ponderación de patrones [24] para reducir en la medida de lo posible los costes computacionales en la última etapa.

Este esquema de ponderación es aplicado para cada una de las clases (como en la anterior etapa) y comienza asignando en la primera iteración un peso de 1 para todos los patrones de la clase objetivo, posteriormente se ordenan de mejor a peor las reglas según el criterio de evaluación siguiente:

$$wWRAcc''(A \to C_k) = \frac{n''(A \cdot C_k)}{n'(C_k)} \cdot \left(\frac{n''(A \cdot C_k)}{n''(A)} - \frac{n(C_k)}{N}\right)$$

Donde $n(C_k)$ es el número de patrones de la clase C_k, $n'(C_k)$ es la suma de los pesos de los patrones de la clase C_k, $n''(A)$ es el sumatorio de los productos de los pesos de todos los patrones con los grados de compatibilidad con el antecedente de las reglas, $n''(A \cdot C_k)$ es el sumatorio de los productos de los pesos de todos los patrones correctamente cubiertos por los grados de compatibilidad con el antecedente de las reglas y N es el número total de patrones.

Este criterio obtiene valores en el rango [-1,1] por lo tanto, cuanto más cerca de 1 se encuentre el resultado de aplicar este criterio una regla, esta será más interesante para la clasificación y por tanto mejor.

Una vez realizada la ordenación de las reglas, se selecciona la mejor de todas y se vuelven a ponderar los patrones cubiertos por ella según la fórmula $w_p = \frac{1}{C_p}$, donde C_p es el recuento de cuantas veces ha sido cubierto el patrón p-ésimo por las reglas seleccionadas.

Este proceso se repite hasta que se cumpla uno de los criterios de parada, estos son: o bien todos los patrones han sido cubiertos más de k_t veces o bien no hay más reglas en la base de reglas.

Etapa 3. Selección genética de reglas y ajuste lateral:

En esta última etapa del proceso de aprendizaje de FARC-HD, una vez aprendidas y seleccionadas las reglas en la etapa 1 y 2, se realiza un proceso evolutivo que permite por un lado optimizar la posición lateral de las funciones de pertenencia y realizar una selección de reglas para poder conseguir un conjunto compacto de reglas de asociación difusas con la mayor precisión en la clasificación. Para realizar el ajuste lateral se introduce el **parámetro de traducción simbólica** α, este parámetro es un valor dentro del intervalo [-0.5,0.5) que expresa el rango de desplazamiento de un valor lingüístico cuando se desplaza entre sus dos valores lingüísticos laterales. Para entender cómo opera α, en la figura 4 podemos observar una partición difusa con 5 valores lingüísticos (S_0-S_4), la traducción simbólica de uno de estos valores y el desplazamiento lateral de la función de pertenencia resultante.

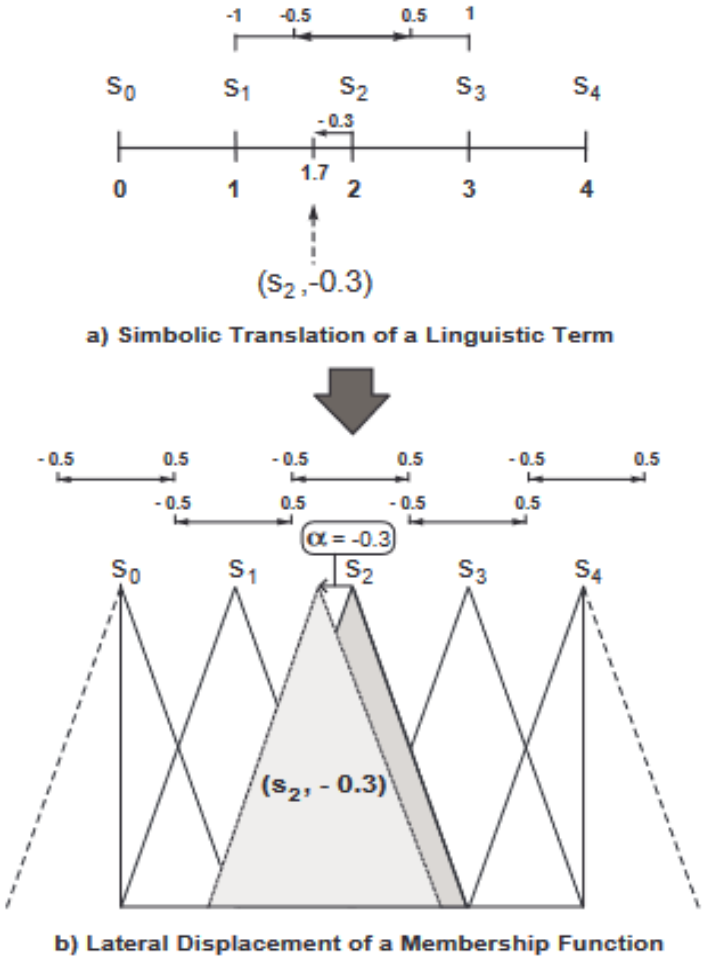

Figura 4: Traducción simbólica de una etiqueta lingüística y desplazamiento lateral de la función de pertenencia implicada

El proceso evolutivo llevado a cabo en esta etapa para la selección de reglas y el ajuste lateral se compone de los siguientes elementos: un modelo genético, la codificación de los genes y cromosoma inicial y la evaluación de los cromosomas.

I. Modelo genético: En FARC-HD el algoritmo genético utilizado es CHC, este algoritmo presenta un buen balance entre exploración y explotación y ha demostrado obtener muy buenos resultados sobre espacios de búsqueda complejos, en este proyecto se van a proponer diferentes alternativas para este algoritmo que se presentarán a continuación.

II. Codificación de los genes y cromosoma inicial: Para poder realizar tanto la selección de reglas como el ajuste lateral es necesario codificar la información de manera que estas dos tareas queden representadas en cada uno de los cromosomas, por ello en FARC-HD se propone un esquema doble de codificación en el que un cromosoma queda determinado por una parte que llamaremos C_s que codificará la información para la selección de reglas y otra parte que llamaremos C_t que codificará la información para el ajuste lateral. La parte C_s del cromosoma sigue una codificación binaria que indica si una regla queda seleccionada (valor 1) o no (valor 0), esta parte se representa mediante un vector de i posiciones donde i es el número de reglas candidatas de tal forma que $C_s = \{C_1, \dots, C_i\}$. La parte C_t del cromosoma por otro lado sigue una codificación real, esta parte codifica los parámetros α de cada partición difusa del problema, de modo que si tuviésemos un problema con n variables y el número de valores lingüísticos para cada variable viniese determinado por $(m^1, \dots, m^n)$ esta parte tendría la forma $C_t = (C_{11}, \dots, C_{1m^1}, C_{21}, \dots, C_{2m^2}, C_{n1}, \dots, C_{nm^n})$. Como resultado de esta doble codificación, un cromosoma quedará determinado por la unión de estas dos partes de la siguiente manera: $C = C_s C_t$. Además, a la hora de generar una nueva población para el modelo genético, un cromosoma especial es incluido. Este cromosoma especial está formado por todos los genes de la parte C_s con valor 1 y todos los genes de la parte C_t con valor 0 se añade a la población inicial con la finalidad de que al menos un individuo de la población utilice todas las reglas candidatas, esto favorece la exploración en todas las regiones del espacio de búsqueda y el aprovechamiento de toda la información disponible.

III. Evaluación de los cromosomas: Para determinar la calidad de un cromosoma se utiliza la siguiente función de fitness:

$$Fitness(C) = \frac{\#\ Hits}{N} - \delta \cdot \frac{NR_{initial}}{NR_{initial} - NR + 1}$$

Donde #Hits es el número de patrones correctamente clasificados, $NR_{initial}$ es el número inicial de reglas candidatas, N es número total de patrones, NR es el número de reglas seleccionadas tras la aplicación del proceso evolutivo y δ es un porcentaje de ponderación determinado por un experto.

En la figura 5 podemos observar el flujo de aprendizaje que realiza el clasificador FARC-HD, dividido en las tres etapas que lo forman.

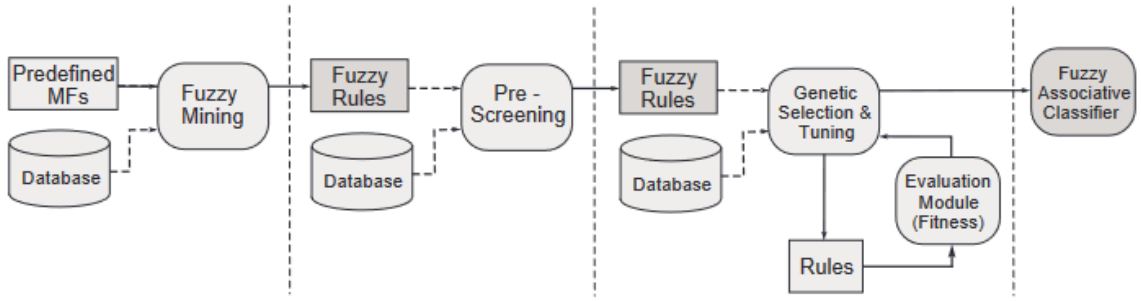

Figura 5: Esquema de aprendizaje modelo FARC-HD

2.6 COMPUTACIÓN EVOLUTIVA E INTELIGENCIA DE ENJAMBRE

Dentro de las técnicas metaheurísticas basadas en poblaciones para resolver problemas de optimización (ver figura 6) [25], podemos distinguir dos categorías principales en función del fenómeno natural en el que inspiran su diseño: los **algoritmos evolutivos** que basan su diseño en la teoría de la evolución de Darwin y los **algoritmos de inteligencia de enjambre** inspirados en el comportamiento colectivo de colonias de animales sociales.

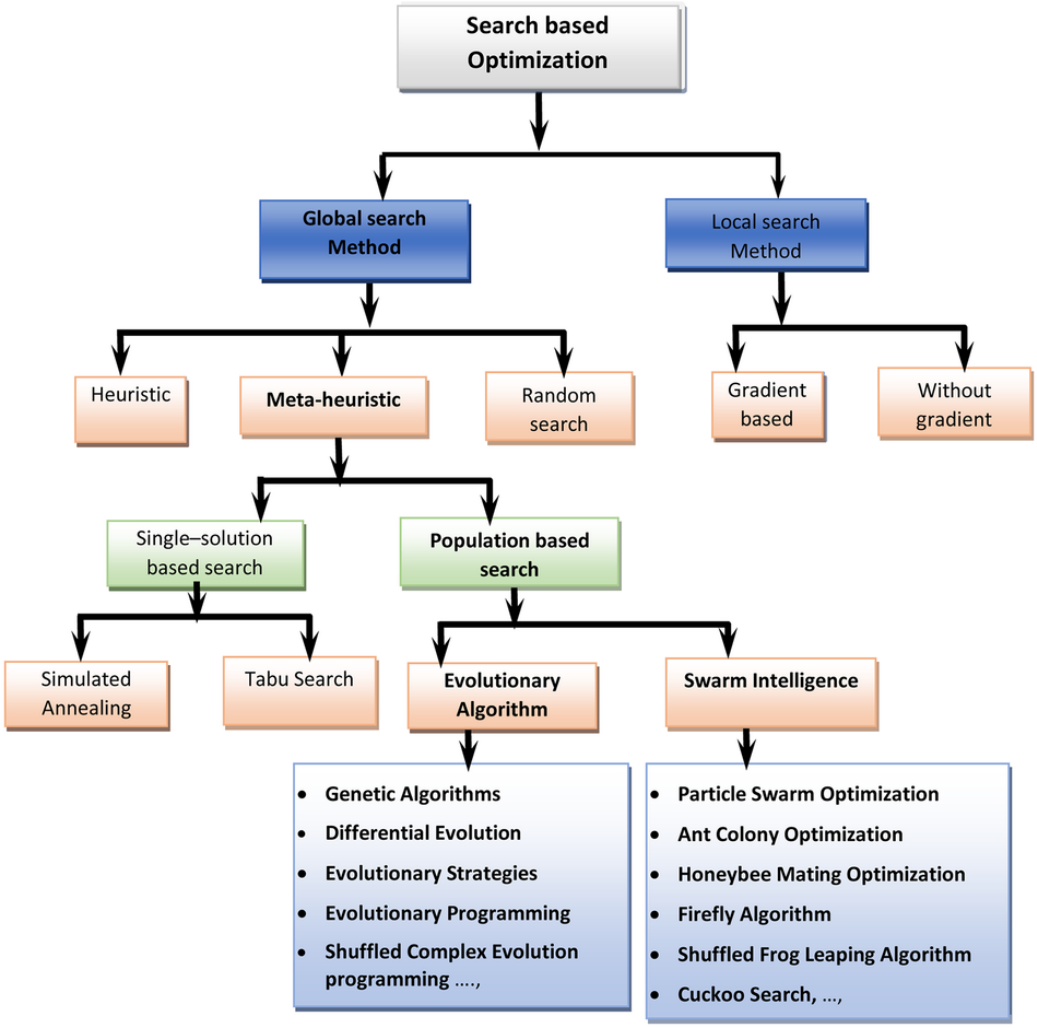

Figura 6: Taxonomía de métodos de optimización

La computación evolutiva (CE) [26] [27], término acuñado en 1991 es un campo de la inteligencia artificial centrado en el desarrollo de **algoritmos evolutivos** (AE) para resolver problemas de búsqueda y optimización computacionalmente difíciles. Surgió en la década de 1950 cuando se empezó a aplicar la teoría evolutiva de Charles Darwin en un intento de simular mediante modelos computacionales los sistemas evolutivos dados en los procesos biológicos y los mecanismos de selección natural. Además, la computación evolutiva toma toda su terminología de los campos de estudio de la genética y la biología. No fue hasta 1960 cuando tres líneas de investigación independientes desarrollaron los tres modelos computacionales más conocidos denominados algoritmos evolutivos en los que se fundamentan todas las investigaciones posteriores, estos son los algoritmos genéticos (AGs), las estrategias de evolución (EEs) y la programación evolutiva (PE). Así mismo, como basan su funcionamiento en los mecanismos de evolución, podemos describir un conjunto de características comunes a todos los AEs:

- Están basados en poblaciones de individuos en las que cada individuo representa una solución en un punto del espacio de todas las posibles soluciones.
- Cada individuo es evaluado conforme una medida de calidad denominada fitness o medida de adaptación.
- Implementan un operador de selección en el que los individuos más adaptados tendrán mayor probabilidad de reproducción y supervivencia en el tiempo que los individuos menos adaptados.
- Utilizan procesos de recombinación y mutación para generar una descendencia similar pero no idéntica a sus progenitores.

Las principales diferencias entre algoritmos evolutivos residen fundamentalmente en cómo son diseñadas e implementadas estas cuatro características básicas. Ya sea el tipo de codificación utilizada para representar a un individuo o que diseño se utilice para construir los operadores de mutación, recombinación y selección y qué importancia se le dé a cada uno de estos operadores, estas elecciones darán como resultado nuevos modelos computacionales con diferentes propiedades.

En este proyecto en concreto vamos a tratar con tres algoritmos dentro de la familia de los algoritmos evolutivos: CHC que es el implementado por defecto en FARC-HD, el algoritmo genético clásico y la evolución diferencial (una variante de los algoritmos evolutivos).

Por otro lado, recientemente la comunidad científica ha empezado a volcar su interés en el estudio del comportamiento de ciertas sociedades de insectos o animales [28]. Dentro de este campo ha sido muy estudiado el movimiento gregario en estas sociedades al que denominamos comportamiento de enjambre con el fin de entender

como interactúan, evolucionan y alcanzan sus objetivos los individuos que lo forman, ya sea a la hora de buscar comida, huir de un depredador o dar caza a una presa.

La **inteligencia de enjambre** o SI por sus siglas en inglés es una rama de la inteligencia artificial que hace uso de este estudio del comportamiento de enjambre y la sociocognición para desarrollar algoritmos capaces de resolver diferentes tareas de optimización [25].

Al igual que en los algoritmos evolutivos, este tipo de algoritmos de optimización están formados inicialmente por una población de individuos habitualmente llamados agentes simples (por su capacidad de percibir su entorno y actuar en consonancia) pero se diferencian de los AEs en que no utilizan operadores evolutivos como la mutación o la recombinación. En los algoritmos SI la población (enjambre) actúa sin supervisión externa de manera autoorganizada donde cada individuo utiliza su entorno y recursos e interacciona de manera local para poder en conjunto dar una respuesta a nivel global.

Para entender el funcionamiento básico de este tipo de algoritmos podemos pensar en todas las posibles soluciones de nuestro problema como un espacio de búsqueda en el que debemos encontrar las mejores (optimización), de modo que un individuo de la población representará un punto en ese espacio, es decir, una solución y este “volará” o se moverá a través del espacio de búsqueda modificando su dirección y velocidad en base a su interacción con los otros individuos de la población y el entorno.

Los algoritmos más conocidos dentro de esta rama de investigación diferenciados entre sí por el sistema social en el que se inspiran son: optimización mediante enjambre de partículas o particle swarm optimization (PSO), optimización de colonias de hormigas o ACO y optimización de apareamiento de abejas melíferas o HBMO.

En este proyecto en concreto trataremos con el algoritmo PSO y sus variaciones.

2.6.1 CHC

CHC es un algoritmo genético no tradicional de la familia de los algoritmos evolutivos desarrollado por Larry J. Eshelman en 1991 [29]. Fue una de las primeras propuestas en introducir un equilibrio entre exploración y explotación del espacio de búsqueda gracias a la inserción de nuevos componentes como el reemplazo elitista, el cruce uniforme HUX, la prevención de incesto o la reinicialización.

Además, CHC prioriza el operador de recombinación sobre el resto de los operadores genéticos y a diferencia de los algoritmos genéticos clásicos no aplica el operador de mutación, en su lugar se aplica la reinicialización o Cataclysmic mutation para introducir diversidad en la población. La selección elitista por otro lado propicia una elevada convergencia del algoritmo, esto en muchas ocasiones puede ser beneficioso, pero en ciertas situaciones puede tender a estancar el algoritmo en mínimos locales, es por ello por lo que el cruce HUX, la prevención de incesto y la reinicialización son introducidos en este algoritmo para elevar la diversidad y consecuentemente la exploración de nuevas regiones del espacio de búsqueda. En la figura 7 se muestra el pseudocódigo del algoritmo CHC.

Algoritmo 1 Pseudocódigo para el algoritmo CHC

Parámetros: tamaño de la población *p*, longitud de cromosoma *L* y umbral de diferencia *d*.

Inicio

$t \leftarrow 0$
$d = L/4$
Inicializar (P(t))
Evaluar (P(t))
Repetir
$t = t + 1$
C(t) = *Seleccionar (P(t-1))*
C'(t) = Recombinar (C(t))
Evaluar (C'(t))
P(t) = Reemplazar (C'(t), P(t-1))
Si *P(t) = P(t-1)*
d--;
Si *d<0*
P(t) = Reinicializar(P(t))
Reset(d)

Hasta *condición_parada*

Fin

Figura 7: Pseudocódigo del algoritmo CHC

Inicialización:

Para iniciar la población del algoritmo CHC se generan *p* individuos aleatoriamente. La parte C_s del cromosoma para la selección de reglas se construirá con valores aleatorios tomados del conjunto {0,1} y la parte C_t para el ajuste lateral se construirá con valores aleatorios en el rango [0,1] y como se ha mencionado anteriormente se introduce un cromosoma especial con todas las reglas seleccionadas y valor 0 en toda la parte C_t con la intención de favorecer la exploración de todas las regiones del espacio de búsqueda.

Selección:

En CHC la selección de los individuos se realiza mediante el método de selección aleatorio de la bajara. Para aplicar este método de selección se inicializa un vector de tamaño *p* donde los valores van desde 1 hasta *p* de tal forma que la posición i-ésima del vector representa el cromosoma i-ésimo de la población. Posteriormente para cada elemento de ese vector se obtiene un valor aleatorio entre 1 y *p* y se intercambia el valor de elemento que se esté tratado por el valor aleatorio generado.

Recombinación:

Debido a que FARC-HD opera con cromosomas formados por una parte binaria y otra parte real, la recombinación original de CHC será modificada para adaptarla a este tipo de cromosomas de modo que se aplicará un operador de cruce diferente para cada una de las partes.

La elección del operador de cruce **HUX o Half Uniform Crossover** nace de la idea de buscar un equilibrio a la hora de combinar a los progenitores para formar nueva descendencia, el problema es que cuantos más genes copiemos del primer progenitor menos genes serán copiados del segundo y viceversa. Para solucionar este problema HUX propone intercambiar exactamente la mitad de los genes no coincidentes de cada progenitor (aleatoriamente escogidos) de manera que se maximice la posibilidad de que dos buenos esquemas, uno de cada progenitor, se combinen en un hijo. Además, garantiza que los descendientes posean la máxima distancia Hamming con sus progenitores lo que propicia la introducción de diversidad (exploración) al generar descendientes muy diferentes de los progenitores. Este esquema de cruzamiento que es el que por defecto aplica CHC, será implementado para generar los dos descendientes de la parte de selección de reglas C_s.

Por otro lado, en la parte de ajuste lateral C_t se aplicará el operador **Parent Centric BLX-α o PCBLX-α**. Este operador está basado en el concepto de vecindad, en concreto en los operadores parent centric, cada cromosoma tiene un rol específico determinado por una diferenciación sexual entre cromosomas masculinos y cromosomas femeninos. Los cromosomas femeninos son utilizados para centrar la

región donde se crearán los descendientes mientras que los masculinos se utilizarán para calcular la extensión de la región (véase figura 8). El funcionamiento de este tipo de cruce también debe ser adaptado para operar en FARC-HD debido a que en su origen PCBLX-α solo genera un descendiente y FARC-HD requiere dos, por ello cada uno de los progenitores tomará ambos roles. Supongamos que $X = (x_1, \dots, x_n)$ e $Y = (y_1, \dots, y_n)$ son los cromosomas que van a ser cruzados donde sus genes toman valores en el intervalo $[a_i, b_i]$, los dos descendientes son generados de la manera que sigue:

- $D_1 = (d_{11}, \dots, d_{1n})$ donde el elemento d_{1i} es escogido aleatoriamente de un intervalo $[l_i^1, u_i^1]$ siendo $l_i^1 = \max\{a_i, x_i - I_i \cdot \alpha\}$, $u_i^1 = \min\{b_i, x_i + I_i \cdot \alpha\}$ e $I_i = |x_i - y_i|$.

- $D_2 = (d_{21}, \dots, d_{2n})$ donde el elemento d_{2i} es escogido aleatoriamente de un intervalo $[l_i^2, u_i^2]$ siendo $l_i^2 = \max\{a_i, y_i - I_i \cdot \alpha\}$, $u_i^1 = \min\{b_i, y_i + I_i \cdot \alpha\}$ e $I_i = |x_i - y_i|$.

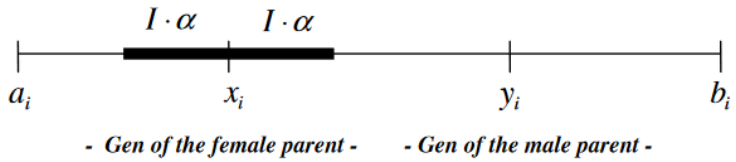

Figura 8: Esquema funcionamiento PCBLX-α

Una vez aplicado el operado correspondiente a cada una de las partes se habrán generado cuatro descendientes, dos de la parte C_t y dos de la parte C_s, combinándolos entre sí obtendremos los cuatro descendientes candidatos. De los cuatro descendientes candidatos se escogerán los dos mejores.

Reemplazo elitista:

En el algoritmo CHC, a diferencia de los algoritmos genéticos clásicos en vez de sustituir la antigua población progenitora por la descendencia que han generado, la descendencia debe competir con sus progenitores por la supervivencia de tal forma que los miembros más aptos serán conservados en la nueva población. Este **reemplazo elitista** selecciona a los N mejores cromosomas entre el conjunto de progenitores y descendientes para permanecer en la población.

Prevención de incesto:

La idea principal de introducir un mecanismo de prevención de incesto en el algoritmo CHC es evitar cruzar individuos similares para obtener mayor diversidad en los descendientes y realizar una primera búsqueda exploratoria del espacio de soluciones, posteriormente las restricciones de este mecanismo irán relajándose de manera que conforme se desarrolle el proceso evolutivo se realice una mayor explotación de aquellas regiones más prometedoras permitiendo que progenitores más similares esta vez sí que se crucen.

Para ello en FARC-HD solo se cruzarán las parejas de progenitores que difieran en un número determinado de genes (**umbral de cruce**) utilizando como medida de diferencia la distancia de Hamming que mide el número de bits que tienen que cambiarse para transformar un progenitor en el otro. Como en FARC-HD tenemos una parte en codificación real será necesaria una transformación previa a binario mediante un código gray con un número fijo de bits por gen (#BITSGENE) para poder aplicar este mecanismo. El umbral de cruce inicial se establece como la distancia máxima posible entre dos individuos dividido por cuatro y se decrementará un $\varphi\%$ de su valor cuando no haya nuevos individuos en la población, dos individuos se cruzarán si su distancia Hamming dividida entre dos es mayor que este umbral.

Reinicialización:

Este mecanismo de reinicio se establece en CHC como alternativa al operador de mutación en el AG tradicional como forma de renovar la población cuando haya sido alcanzada una convergencia prematura hacia óptimos locales o la búsqueda se haya estancado. En ese caso el mejor cromosoma de la población es conservado tras el reinicio y los individuos restantes se generarán aleatoriamente. Este mecanismo de reinicio, también llamado **cataclysmic mutation** por su drástica forma de introducir nueva diversidad en la población se aplica en FARC-HD cuando el umbral de cruce es inferior a cero. Tras el reinicio este umbral es nuevamente reestablecido a su valor inicial.

Criterio de parada:

Los criterios de parada que habitualmente se implementan en CHC son haber alcanzado el máximo de iteraciones establecidas por el usuario, haber realizado varios reinicios de la población sin mejora (el fitness del mejor individuo no cambia) o haber alcanzado un óptimo que se ajuste a nuestras restricciones. En nuestro caso, en FARC-HD implementamos los criterios de haber realizado varios reinicios de la población sin mejora y el haber alcanzado un número máximo de cromosomas evaluados establecido por el usuario.

2.6.2 ALGORÍTMO GENÉTICO CLÁSICO

Los algoritmos genéticos o AGs son un tipo de algoritmos de búsqueda dentro de la familia de los algoritmos evolutivos que realizan búsquedas adaptativas y heurísticas basadas en la teoría de la evolución de Darwin, la selección natural y la genética [30]. Este tipo de algoritmos han sido ampliamente utilizados en la resolución de problemas de optimización del mundo real gracias a su capacidad de realizar búsquedas en espacios de soluciones muy grandes y a su robustez frente a otros métodos de la inteligencia artificial. Además, a pesar de que este tipo de algoritmo es aleatorio, utiliza la información aprendida sobre un espacio de búsqueda para ir orientando la búsqueda hacia lugares del espacio más prometedores.

Un algoritmo genético tradicional comienza inicializando un conjunto de soluciones candidatas que formarán la población inicial y evaluando su calidad mediante una función de fitness adaptada al tipo de problema que se esté resolviendo. Esta población inicial evoluciona creando nuevas generaciones a través del cruzamiento entre individuos bien adaptados generando nuevos descendientes que pueden mutar. Los descendientes sustituyen mediante algún mecanismo a los progenitores en las generaciones siguientes en un proceso iterativo. En la figura 9 se muestra el pseudocódigo para el algoritmo genético clásico.

Algoritmo 2 Pseudocódigo para el algoritmo genético clásico

Parámetros: tamaño de la población *p*, probabilidad de cruce *Pc,* probabilidad de mutación *Pm*.
Inicio
    $t \leftarrow 0$
    Inicializar (P(t))
    Evaluar (P(t))
    Mientras *condición_parada ≠ Verdadero* **Hacer**
        $t = t + 1$
        C(t) = *Seleccionar (P(t-1))*
        C'(t) = Recombinar (C(t), Pc)
        C''(t) = Mutar(C'(t), Pm)
        P(t) = Reemplazar (C''(t), P(t-1))
        Evaluar (P(t))
    Fin_mientras

Fin

Figura 9: Pseudocódigo del algoritmo genético clásico

Inicialización y Evaluación:

La inicialización de la población se realiza de la misma manera que en el algoritmo CHC y la evaluación de los individuos de la población se realizará siguiendo el método de evaluación que utiliza FARC-HD por defecto explicado en la tercera etapa del apartado **2.5.2.**

Selección:

El operador de selección en los algoritmos genéticos trata de simular el proceso de selección natural, en este proceso se escogen de manera aleatoria de entre todos los individuos de la población aquellos que se cruzarán y producirán descendencia. Aun que todos los individuos tienen posibilidades de ser escogidos, aquellos mejor adaptados (mejor fitness) tendrán mayor probabilidad de cruzarse que aquellos peor adaptados. En este proyecto se han implementado, además de la **selección aleatoria** utilizada en CHC, tres métodos ampliamente conocidos de selección: **método de la ruleta**, **método del torneo** y **método del ranking**.

- **Método de la ruleta:** En este método se asigna una probabilidad de selección proporcional al valor del fitness de cada individuo. Supongamos que tenemos una población de k individuos (cromosomas) C = $\{C_1, \dots, C_k\}$ y la función de fitness positiva f, la probabilidad de escoger al cromosoma i-ésimo viene determinada por:

$$P(C_i) = \frac{f(C_i)}{\sum_{j=1}^{k} f(C_j)}$$

  Una vez obtenida la probabilidad de selección para cada uno de los individuos de la población, se realiza un particionamiento del intervalo unidad [0,1] en subintervalos proporcionales a estas probabilidades de selección de forma que si pensamos en el intervalo unidad como un círculo (uniendo los extremos) obtendremos una ruleta en la que cada partición i corresponde al subintervalo asociado al individuo C_i. Una vez generada esta especie de ruleta, para seleccionar un progenitor bastará con generar un número al azar p en el intervalo [0,1] (hacer girar la ruleta) y devolver el individuo perteneciente al subintervalo que incluye a p. Este método favorece que un individuo pueda ser escogido múltiples veces como progenitor, como se puede observar en la figura 10, el individuo 4 presenta la mayor partición en la ruleta y por tanto será elegido más veces. Los principales inconvenientes de este método es la convergencia prematura (si en la población hay individuos con un fitness mucho mejor que el resto) y la convergencia lenta del algoritmo (si todos los individuos poseen un fitness muy similar).

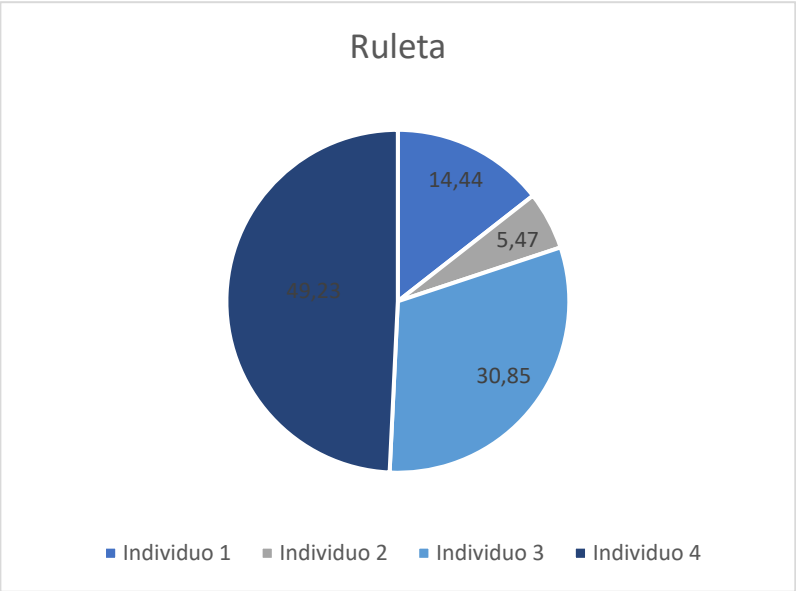

Figura 10: Probabilidad de selección ruleta

- **Método del torneo:** Este método es uno de los más utilizados en los AGs por su sencillez. La idea principal de este método es hacer competir a los individuos de la población mediante torneos de pocos participantes e ir escogiendo a los ganadores. En concreto se escogen aleatoriamente k individuos de la población candidatos a ser progenitores y se orquesta un torneo entre ellos seleccionando el ganador en base a la función de fitness, este proceso se repite tantas veces como número de progenitores se desee generar. Dentro del método del torneo podemos distinguir dos tipos: torneo sin reemplazamiento y con reemplazamiento.

- **Método del ranking:** La idea de este método es ordenar todos los individuos de la población de acuerdo con su fitness y asociar a cada uno de ellos una probabilidad de selección que depende de su posición en el ranking. Para aplicar este método, en primer lugar, se ordenan los individuos de la población decrecientemente en base a su valor en la función de fitness de tal forma que la posición 0 del ranking corresponda al individuo del mejor fitness, posteriormente se asigna a cada individuo un factor en función de su posición calculado mediante la expresión:

$$h(C_i) = N\left[-(N-1) * \frac{Rango(C_i)}{N}\right]$$

Siendo N el tamaño de la población y $Rango(C_i)$ la posición del individuo en el ranking. Por último, se calculan la probabilidad de selección para cada individuo mediante la expresión:

$$p(C_i) = \frac{h(C_i)}{\sum_{i=1}^{N} h(C_i)}$$

Recombinación:

La recombinación de los progenitores en este proyecto se realiza de la misma forma que en CHC, de manera que el cruzamiento queda adaptado para la codificación doble de FARC-HD. Para la parte binaria de selección de reglas C_s se utiliza el cruzamiento HUX y para la parte de ajuste lateral C_t se utiliza el cruzamiento PCBLX-α. Pero a diferencia del algoritmo CHC que se utilizaba la distancia Hamming, en el algoritmo genético clásico introducimos el parámetro de probabilidad de cruzamiento (Pc) escogida por el usuario. Por lo tanto, dos progenitores se cruzarán si un número *p* escogido al azar en el intervalo [0,1] está dentro de esta probabilidad de cruzamiento.

Mutación:

El operador de mutación se encarga de introducir diversidad en la población mediante la alteración aleatoria de algunos genes de los individuos que la forman. Este operador se aplica a cada uno de los descendientes generados en la recombinación en base a una probabilidad de mutación (Pm) determinada por el usuario. En primer lugar, se recorren los descendientes, generando para cada uno de ellos un número aleatorio *p* en el intervalo [0,1]. Si *p* entra dentro de la probabilidad de mutación entonces se aplicará la mutación al descendiente. Haciendo uso de la misma probabilidad de mutación, se recorrerán los genes del descendiente a mutar generando de nuevo, para cada uno de ellos, un número aleatorio *p.* Si este número entra dentro de la probabilidad de mutación entonces el gen sufrirá una mutación. Para mutar los genes de la parte binaria del tipo de cromosomas utilizados en FARC-HD, en este proyecto se aplicará la operación de negación de modo que si el gen tuviese valor 0 pasaría a 1 y viceversa. Por otro lado, para la parte real se aplicará el operador de **reseteo aleatorio,** en el que el valor del gen será cambiado por otro valor escogido aleatoriamente dentro del intervalo de valores posibles, en nuestro caso dentro del intervalo [0,1] para todos los genes reales del cromosoma.

Reemplazo:

Una vez obtenidos los descendientes y aplicado el operador de mutación sobre ellos, debemos formar la nueva población de individuos para la siguiente iteración del algoritmo. En este proyecto se ha seguido un modelo de algoritmo genético generacional, es decir, en cada iteración se crea una población completa nueva que reemplaza directamente a la antigua. Las estrategias de reemplazo utilizadas en este proyecto, siguiendo el enfoque generacional son tres: un reemplazo clásico al que hemos denominado *generational replacement (GR),* y dos reemplazos elitistas, uno

denominado *elitist replacement (ER)* y otro denominado *best k parents replacement (BKPR).*

- ***GR:*** En esta estrategia en cada iteración la población es sustituida por los descendientes generados para formar la población de la iteración siguiente.
- ***ER:*** En esta estrategia, se une la población antigua y la nueva formada por los descendientes y se seleccionan de todos los individuos los N mejores siendo N el tamaño de la población.
- ***BKPR:*** En esta estrategia la nueva población estará formada por los N-k descendientes y los k progenitores con mejor fitness siendo k el número de progenitores a conservar en la siguiente iteración y N el tamaño de la población.

Criterio de parada:

Para poder comparar el algoritmo CHC con el algoritmo genético implementado utilizaremos el mismo criterio de parada: haber alcanzado un número máximo de cromosomas evaluados.

2.6.3 DIFERENTIAL EVOLUTION

Diferential evolution o DE es un tipo de algoritmo dentro de la familia de los algoritmos evolutivos propuesto por Rainer Storn en 1997 [31] [32]. Está enfocado en la resolución de problemas de optimización global en espacios de búsqueda continuos y tiene como particularidad la elección del operador genético de mutación como operador principal y más importante. Una de las principales ventajas del algoritmo DE es su capacidad de obtener buenos resultados a pesar de ser un algoritmo muy sencillo. Además, este algoritmo requiere pocos parámetros y los que utiliza son muy robustos, lo que hace muy sencillo su ajuste.

A diferencia de los algoritmos genéticos, en DE la población de individuos está compuesta por N vectores de parámetros de D dimensiones, donde D es el número de variables del problema que estemos tratando. En cada iteración del proceso evolutivo, para cada vector progenitor de la población (también llamados vectores objetivo), se genera un vector mutante (también llamado vector dotante) por medio de la mutación diferencial utilizando varios vectores escogidos aleatoriamente de la población y diferentes del vector objetivo. Por último, se crea la población de vectores candidatos (llamados vectores de prueba) mediante el cruce de los vectores objetivo con sus correspondientes vectores donantes. Si el vector de prueba obtenido es mejor que el vector objetivo, este lo sustituirá en la población de la siguiente iteración. En la figura 11 se muestra el pseudocódigo para el algoritmo DE.

Algoritmo 3 Pseudocódigo para el algoritmo DE

Parámetros: tamaño de la población *p*, probabilidad de cruce *Cr,* factor de escalado *F*.
Inicio
$t \leftarrow 0$
Inicializar (P(t))
Evaluar (P(t))
Mientras *condición_parada ≠ Verdadero* **Hacer**
$t = t + 1$
V(t) = *Mutar(P(t-1))*
U(t) = Recombinar (V(t), P(t-1))
Evaluar (U(t))
P(t) = Reemplazar (U(t), P(t-1))
Fin_mientras
Fin

Figura 11: Pseudocódigo del algoritmo DE

Inicialización y Evaluación:

La inicialización de la población se realiza de la misma manera que en el algoritmo CHC y la evaluación de los individuos de la población se realizará siguiendo el método de evaluación que utiliza FARC-HD por defecto explicado en la tercera etapa del apartado **2.5.2.**

Mutación diferencial:

En los algoritmos DE se aplica el operador conocido como **mutación diferencial**. Este operador, a diferencia de la mutación tradicional, no muta los genes de ciertos individuos, sino que genera para cada uno de los vectores objetivo de la población, nuevos individuos llamados vectores donantes. Estos vectores donantes se construyen añadiendo a un vector base de la población un número determinado de vectores diferencia ponderados mediante el factor de escalado F.

El factor F es un número perteneciente al intervalo (0,1] que controla la amplificación de la variación diferencial, este factor es ajustado para encontrar un buen balance entre la exploración y la explotación del espacio de búsqueda. Un valor alto en este factor introduce diversidad en la población evitando la convergencia prematura y propiciando una mayor exploración. En cambio, un valor bajo fomentara la explotación de las regiones más prometedoras.

En función del vector base que es escogido para formar los donantes y del número de vectores diferencia utilizados en su construcción podemos distinguir varios modelos de mutación diferencial. Estos modelos son categorizados mediante la notación

DE/x/y, donde x representa el vector base utilizado e y el número de vectores diferencia considerados.

Si suponemos una población de N vectores objetivo formados por D parámetros de la forma $X_i = \{X_{1,i}, \dots, X_{D,i}\}\ \ \forall i = 1, \dots, N$, los enteros mutuamente excluyentes r_1^i, r_2^i, r_3^i, r_4^i y r_5^i escogidos aleatoriamente en el intervalo [1,N] y diferentes de i y el vector X_{best} como el vector con mejor fitness de la población, podemos describir las expresiones empleadas para construir la población de vectores donante $V_i = \{V_{1,i}, \dots, V_{D,i}\}\ \ \forall i = 1, \dots, N$ de los modelos de mutación diferencial utilizados en este proyecto como:

- **DE/rand/1:** $V_i = X_{r_1^i} + F \cdot \left(X_{r_2^i} - X_{r_3^i}\right)$
- **DE/best/1:** $V_i = X_{best} + F \cdot (X_{r_1^i} - X_{r_2^i})$
- **DE/current-to-best/1:** $V_i = X_i + F \cdot (X_{best} - X_i) + F \cdot (X_{r_1^i} - X_{r_2^i})$
- **DE/rand/2:** $V_i = X_{r_1^i} + F \cdot \left(\left(X_{r_2^i} - X_{r_3^i}\right) + \left(X_{r_4^i} - X_{r_5^i}\right)\right)$
- **DE/best/2:** $V_i = X_{best} + F \cdot \left(\left(X_{r_1^i} - X_{r_2^i}\right) + \left(X_{r_3^i} - X_{r_4^i}\right)\right)$

En nuestro caso, debido a que en FARC-HD los cromosomas siguen una codificación doble, es necesario adaptar estos modelos de mutación diferencial de modo que puedan ser aplicados a una codificación binaria. Para la parte real de nuestros cromosomas, bastará con calcular las expresiones anteriores y comprobar que la parte real del vector donante generado es válida. Consideraremos que esta parte es válida si cada uno de sus genes toma valores dentro de su rango de valores posibles, en nuestro caso en el intervalo [0,1] para todos nuestros genes reales, de no ser así, aquellos genes que no entren dentro del intervalo de valores posibles serán ajustados a sus límites. Con el fin de adaptar la mutación diferencial para la parte binaria de nuestros cromosomas se han utilizado las estrategias propuestas por T. Gong y A. L. Tuson [33]. Suponiendo tres vectores aleatorios de la población X_1, X_2 y X_3 con los que vamos a formar el vector donante, estas estrategias son:

- **Restricted-change DE mutation (M_{res}):** En esta primera estrategia, consideramos cada gen del cromosoma como una única dimensión, de forma que la distancia (diferencia) entre dos vectores para cada dimensión es 0 o 1. Por ejemplo, si consideramos los vectores $X_2 = (1,1,0,1,1)$ y $X_3 = (0,1,1,1,0)$, la distancia entre ellos resultará como $Distance = (1,0,1,0,1)$. Siguiendo el arquetipo de la mutación diferencial, para interpretar la diferencia escalada que se añade a un vector base, en esta estrategia se introduce la siguiente expresión para el cálculo del vector donante:

$$D_j(donante, X_1) = \begin{cases} 1, & si\ rand_j < F \ \wedge\ D_j(X_2, X_3) = 1 \\ 0, & en\ otro\ caso \end{cases}$$

En esta expresión, la diferencia entre los vectores X_2 y X_3 y el factor de escalado F son utilizados para calcular la probabilidad de producir un cambio en el gen de la posición j del vector base X_1 para formar el vector donante. Es decir, para cada posición j del vector donante, este tomará el mismo valor que el vector base X_1 si la distancia en esa posición entre los vectores X_2 y X_3 es 0 (mismo valor), y tomará el valor contrario a X_1 si la distancia entre X_2 y X_3 es 1 y además F es mayor que un cierto número aleatorio escogido en el intervalo [0,1].

- **Any-change DE mutation (M_{any}):** En esta estrategia, consideramos todos los genes del cromosoma como una misma dimensión, de forma que la distancia entre dos vectores se reduce a la distancia Hamming. De tal modo que, siguiendo con el ejemplo anterior la distancia entre X_2 y X_3 resultaría en $d = 3$. Si escalamos esta distancia mediante el factor F de modo que $d' = d \cdot F$, y siendo $(int)\ d'$ e $(int)\ d' + 1$ los enteros más cercanos a d' la distancia entre el vector donante y el vector base X_1 vendría determinada por la siguiente expresión:

$$D(donante, X_1) = \begin{cases} (int)d' + 1, & si\ rand < d' - (int)d' \\ (int)d', & en\ otro\ caso \end{cases}$$

Esta expresión en base al número aleatoriamente escogido en el intervalo [0,1] $rand$, nos indicará el número de cambios (distancia) que deberemos aplicar al vector base para producir el vector donante. Estos cambios son producidos en genes aleatorios del vector y son mutuamente excluyentes.

Recombinación:

Una vez realizada la mutación diferencial, el siguiente paso en el algoritmo DE es aplicar el operador de recombinación o cruce. Este operador cruza la población de vectores objetivo con la población de vectores donante para formar la población de vectores candidatos o vectores de prueba. Este operador hace uso del parámetro Cr que determina la probabilidad de cruzamiento. En este proyecto se han implementado dos cruces ampliamente utilizados en DE:

- **Cruce binomial:** El vector de prueba U_i es formado a partir del vector objetivo X_i y el vector donante V_i aplicando la siguiente expresión:

$$U_{j,i} = \begin{cases} V_{j,i}\,, & si\ \ (rand_j[0,1] \leq Cr \text{ ó } j = j_{rand}) \\ X_{j,i}\,, & en\ otro\ caso \end{cases}$$

Donde j_{rand} es un índice aleatorio en [1, D] que asegura que $U_i \neq X_i$

- **Cruce exponencial:** El vector de prueba U_i es formado a partir del vector objetivo X_i y el vector donante V_i aplicando la siguiente expresión:

$$U_{j,i} = \begin{cases} V_{j,i}\,, & para\ j = \ \langle n\rangle_D, \langle n+1\rangle_D, \dots, \langle n+L-1\rangle_D \\ X_{j,i}\,, & para\ todas\ las\ demás\ j \in [1, D] \end{cases}$$

Donde $\langle\ \ \rangle_D$ es una función módulo con módulo D, n es un número escogido aleatoriamente en el intervalo [1, D] que indica el punto de partida a partir del cual se van a intercambiar los valores con el vector donante y L es un número escogido aleatoriamente en [1, D] que indica el número de elementos que el vector de prueba toma del donante. Si al realizar el cruce se llega al final del vector, este continuará por el principio hasta haber copiado los L elementos.

Reemplazo:

Por último, una vez obtenida la población candidata de vectores de prueba, se aplica el operador de reemplazo. Este operador realiza una comparación entre la población de vectores objetivo y sus correspondientes vectores de prueba de modo que si el vector de prueba posee un valor en la función fitness mejor que el vector objetivo dicho vector lo reemplazará en la próxima generación, en caso contrario el vector objetivo será conservado. Una vez aplicado el reemplazo, la población resultante para la próxima generación tendrá el mismo tamaño que la población inicial.

Criterio de parada:

Para poder comparar el algoritmo CHC con el algoritmo DE implementado utilizaremos el mismo criterio de parada: haber alcanzado un número máximo de vectores evaluados.

2.6.4 PARTICLE SWARM OPTIMIZATION

Particle swarm optimization o PSO es un método de optimización perteneciente a la familia de los algoritmos de inteligencia de enjambre [33][34]. Fue introducido por primera vez en 1995 por J. Kennedy y R. Eberhart quienes

desarrollaron una metodología basada en un enjambre de partículas para la optimización de funciones no lineales. PSO, en concreto se inspira en el comportamiento social del vuelo de bandadas de pájaros en busca de maíz.

En PSO un conjunto de partículas llamadas agentes simples son colocados en el espacio de búsqueda del problema que queremos resolver, cada agente representa un punto en ese espacio además de una solución del problema. A continuación, este conjunto de agentes (enjambre) se mueven a través del espacio de búsqueda determinando su movimiento mediante su conocimiento personal y colectivo, es decir, realizan un intercambio de información entre ellos que les permite modificar su dirección para moverse hacia áreas prometedoras del espacio.

A diferencia de los algoritmos evolutivos, en PSO la población inicial es mantenida a lo largo de todo el proceso iterativo del algoritmo y no son utilizados ni operadores de cruce, ni mutación ni reemplazo. Por otro lado, PSO es capaz de converger rápidamente a una buena solución siendo un buen candidato para nuestro proyecto, además la implementación es muy sencilla y en la literatura existen multitud de variantes, lo que lo hace a PSO un algoritmo muy versátil. En la figura 12 se muestra el pseudocódigo para el algoritmo PSO.

Algoritmo 4 Pseudocódigo para el algoritmo PSO

Parámetros: tamaño de la población *p*, ratio de aprendizaje para la componente cognitiva φ_1, ratio de aprendizaje para la componente social φ_2, parámetro de ajuste de inercia α.

Inicio

$t \leftarrow 0$

Inicializar (P(t))

Mientras *condición_parada ≠ Verdadero* **Hacer**

$t = t + 1$

EvaluarPartículas (P(t))

ActualizarPartículas(P(t), $\varphi_1,\ \varphi_2, \alpha$*)*

Fin_mientras

Fin

Figura 12: Pseudocódigo del algoritmo PSO

Inicialización:

Cada agente simple que compone una población en PSO está compuesto por un vector X que indica su posición actual en el espacio, un vector $pBest$ que almacena la localización de la mejor posición encontrada hasta el momento, un vector V que almacena el gradiente según el cual se moverá la partícula, el valor de fitness de la posición actual en la que se encuentre el agente ($x_{fitness}$) y el valor de fitness

correspondiente a su mejor posición encontrada en el trascurso del algoritmo ($p_{fitness}$).

Para iniciar una población adaptando PSO a la doble codificación utilizada en FARC-HD se sigue el mismo procedimiento que en el algoritmo CHC para inicializar las posiciones de cada agente de manera aleatoria y se añade la inicialización de los vectores que almacenan la velocidad: un vector V para la parte real del cromosoma y un vector VR para la parte binaria, ambos vectores con valores 0 en todas sus posiciones. Esta primera inicialización, como podemos ver en figura 13, intenta distribuir partículas en todo el espacio de búsqueda de forma que todas las regiones sea susceptibles a ser exploradas.

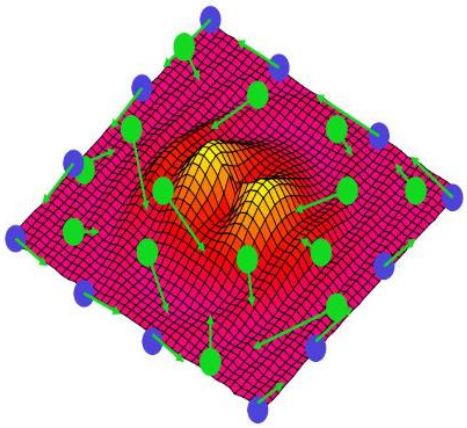

Figura 13: Distribución inicial de partículas

Evaluación de partículas:

En este paso del algoritmo, para cada uno de los agentes de la población, se evalúa su posición actual mediante la misma función de fitness que en CHC. Posteriormente la información del agente es actualizada de la siguiente manera:

- $\boldsymbol{Si}\ x_{fitness} \geq p_{fitness}\ \boldsymbol{entonces}\ pBest = X\ \boldsymbol{y}\ p_{fitness} = x_{fitness}$

En el caso de haber optado por un modelo global de PSO (que se explicará en el siguiente apartado), también se actualizará la información de la mejor posición alcanzada por un agente en el transcurso del algoritmo $gBest$ y su correspondiente fitness $g_{fitness}$ de la siguiente manera:

- $\boldsymbol{Si}\ p_{fitness} \geq g_{fitness}\ \boldsymbol{entonces}\ gBest = pBest\ \boldsymbol{y}\ g_{fitness} = p_{fitness}$

Actualización de partículas:

Una vez evaluados todos los agentes de la población, el siguiente paso en el proceso del algoritmo PSO es mover cada uno de ellos a una nueva posición del espacio de búsqueda. Para llevar a cabo este movimiento, es necesario en primer lugar, actualizar su velocidad en función del conocimiento personal y colectivo adquirido por el agente en el primer paso del algoritmo y, en segundo lugar, actualizar su posición en base a esta velocidad.

La expresión utilizada en este proyecto para actualizar el vector de velocidad V de cada agente i de la población y cada dimensión d, es la siguiente:

$$V_{id}(t+1) = \omega \cdot V_{id}(t) + \varphi_1 \cdot rand() \cdot (pBest_{id} - X_{id}) + \varphi_2 \cdot rand() \cdot (Z - X_{id})$$

Esta expresión se compone de cuatro elementos. En primer lugar, se parte de la velocidad que llevaba hasta el momento el agente ($V_{id}(t)$) y en base al conocimiento adquirido por este, se agregan dos elementos más, una componente cognitiva ($\varphi_1 \cdot rand \cdot (pBest_{id} - X_{id})$) y otra componente social ($\varphi_2 \cdot rand \cdot (Z - X_{id})$). La **componente cognitiva** trata de modificar la velocidad para que el agente vuelva al punto en el que más cerca estuvo de un óptimo. Por otro lado, la **componente social** trata de modificar la velocidad para que el agente vuelva al punto donde se encontraba el agente de la población que más cerca estuvo de un óptimo.

En base a los valores que asignemos para las **ratios de aprendizaje** φ_1 y φ_2, obtendremos algoritmos de PSO con diferentes comportamientos, algunos de estos modelos son:

- **Modelo completo:** $\varphi_1, \varphi_2 > 0$
- **Modelo sólo cognitivo:** $\varphi_1 > 0 \; y \; \varphi_2 = 0$
- **Modelo sólo social:** $\varphi_2 > 0 \; y \; \varphi_1 = 0$

Dependiendo de qué agente Z tomemos para formar la componente social, podemos distinguir dos modelos diferentes de PSO utilizados en este proyecto: un **modelo global** y un **modelo local**. Si escogemos al agente Z como $gBest$, es decir, como el agente con mayor fitness a lo largo de todo el transcurso del algoritmo, tendremos un modelo global, en cambio, si escogemos al agente Z como $lBest$, es decir, el agente con mayor fitness de un vecindario restringido, tendremos un modelo local.

Para evitar que la velocidad de los agentes crezca rápidamente, lo que produciría una disminución del rendimiento del algoritmo, se introduce el cuarto componente, un **parámetro de inercia** ω que permite encontrar un buen balance entre la exploración y la explotación. El parámetro ω comienza con valor 1.0 y se va reduciendo gradualmente con el paso del tiempo mediante la expresión:

$$\omega = \left(1 - \left(\frac{t}{t_{max}}\right)^{\alpha}\right)$$

Donde t es la iteración actual, t_{max} es el número de iteraciones máximo del algoritmo y α es un número en el intervalo $(0, \infty)$ fijado por el usuario que regula el ritmo en el que decrece ω. Un valor bajo de α hará que el parámetro ω decrezca rápidamente propiciando una búsqueda más localizada mientras que un valor alto minimizará el efecto de ω propiciando una búsqueda diversificada por más tiempo.

La actualización de la velocidad es el proceso más importante del algoritmo PSO, realiza una función similar a los operadores de cruce y mutación de los algoritmos evolutivos. Mediante la expresión anteriormente descrita, se mantienen ciertas características del individuo y otras se adquieren nuevas de las mejores soluciones (cruce) y, se inserta un componente que permite introducir diversidad en la población (mutación).

Una vez actualizada la velocidad, cada uno de los agentes del enjambre es movido a su nueva posición en el espacio. En nuestro caso, debemos adaptar la expresión para funcionar bajo una codificación doble, por ello, para la parte real utilizaremos la siguiente expresión:

$$X_i(t+1) = X_i(t) + V_i(t)$$

Y para la parte binaria utilizaremos una sigmoide de la forma:

$$Sigmoid(V_{id}) = \frac{1}{1 + e^{-V_{id}}}$$

De forma que cada dimensión del vector posición es actualizada mediante la expresión:

$$X_{id}(t+1) = \begin{cases} 1, & si\ rand() < Sigmoid(V_{id}(t+1)) \\ 0, & en\ otro\ caso \end{cases}$$

Siendo rand() un número aleatorio en el intervalo [0,1].

Al igual que en DE, una vez actualizada la posición para la parte real de cada uno de los agentes del enjambre, es necesaria una comprobación de la validez de las soluciones obtenidas, de manera que en el caso de que el vector de posición tome valores fuera del rango de valores posibles, estos sean ajustados a sus extremos.

Criterio de parada:

Para poder comparar el algoritmo CHC con el algoritmo PSO implementado utilizaremos el mismo criterio de parada: haber alcanzado un número máximo de vectores evaluados.

3 EXPERIMENTACIÓN Y RESULTADOS

En esta sección se van a presentar las diferentes pruebas que se han realizado, los resultados que se han obtenido y un estudio comparativo entre el algoritmo CHC y los algoritmos propuestos en este proyecto. Todos los resultados no incluidos en esta memoria se pueden consultar en el repositorio https://github.com/inigosisniega/TFG-experimental-results.git.

3.1 MARCO EXPERIMENTAL

La experimentación de este proyecto se ha realizado mediante un script escrito en el lenguaje de programación pyhton, desarrollado en el entorno informático interactivo Jupyter Notebook. Este script realizará una ejecución de los modelos genéticos implementados en Java, a través de llamadas reiteradas de la función *run()* de la librería *subprocess* [35], alterando sus hiperparámetros sobre un conjunto de datasets.

Para analizar el rendimiento de los modelos genéticos propuestos en este proyecto, estos serán evaluados sobre un total de 12 conjuntos de datos del mundo real extraídos del repositorio de datasets KEEL [36]. En la tabla 1 se resumen las principales características de cada dataset, mostrando para cada uno de ellos su nombre, el número de variables que lo forman, el número de variables que hay por cada tipo de dato (Real/Entero/Nominal), el número de clases utilizadas para la clasificación y el número de patrones o instancias recopilados. Además, en este proyecto se trabajará con conjuntos de datos completos, por ello, en ninguno de los datasets existirán patrones con valores faltantes y en el caso del dataset Wisconsin que, sí que los tiene, se trabajará con su versión libre de estos.

Asimismo, el repositorio KEEL permite obtener estos datasets ya particionados mediante un modelo de validación cruzada estratificada de 5 particiones, que será el modelo que utilizaremos en este proyecto con objeto de poder validar nuestro clasificador y medir su nivel de generalización cuando se enfrenta a nuevos datos. El modelo SCV realiza un particionamiento del conjunto de datos en 5 particiones, cuatro de estas particiones serán utilizadas para formar el conjunto de entrenamiento (80%) y la partición restante será utilizada como el conjunto de prueba (20%). Este proceso se repetirá 5 veces, tomando cada vez una de las particiones como conjunto de prueba y finalmente se realizará la media aritmética de los resultados de las cinco particiones para obtener el resultado para el conjunto de datos.

Tabla 1: DATASETS UTILIZADOS EN LA EXPERIMENTACIÓN Y SUS CARACTERÍSTICAS

Nombre	Variables	(R/I/N)	Clases	Patrones
Pima	8	(8/0/0)	2	768
Wisconsin	9	(0/9/0)	2	683
Magic	10	(10/0/0)	2	19020
Ring	20	(20/0/0)	2	7400
Twonorm	20	(20/0/0)	2	7400
Haberman	3	(0/3/0)	2	306
Phoneme	5	(5/0/0)	2	5404
Balance	4	(4/0/0)	3	625
Ecoli	7	(7/0/0)	8	336
Page-blocks	10	(4/6/0)	5	5472
Bupa	6	(1/5/0)	2	345
Tae	5	(0/5/0)	3	151

Por otro lado, para poder medir el rendimiento de cada uno de los modelos evaluados en la fase de experimentación de este proyecto, utilizaremos la métrica de accuracy rate o tasa de precisión que representa el porcentaje de ejemplos correctamente clasificados con respecto al total de ejemplos clasificados. El tiempo de ejecución también será comparado entre los modelos con objeto de averiguar que alternativa hace un uso más eficiente de este recurso, así como la convergencia de cada uno de ellos.

Con el propósito de comparar solamente la tercera etapa de selección de reglas y ajuste lateral del clasificador FARC-HD, la configuración de los hiperparámetros utilizados en las dos primeras etapas del clasificador y ciertos parámetros globales se mantendrán invariantes en toda la experimentación (ver tabla 2).

Tabla 2: CONFIGURACIÓN DE PARÁMETROS CLASIFICADOR FARC-HD

Hiperparámetros
Número de etiquetas lingüísticas = 5
Soporte mínimo = 0.05
Confianza mínima = 0.8
Profundidad máxima = 3
K = 2
Número máximo de evaluaciones = 20000
Tamaño de la población = 50
α = 0.02
Bits por gen = 30
Tipo de inferencia = combinación aditiva

3.2 FARC-HD IMPLEMENTANDO EL ALGORITMO CHC

Con el fin de comparar el modelo FARC-HD implementando su algoritmo estándar CHC con el resto de las propuestas realizadas en este proyecto, únicamente se realizará una ejecución de este sobre cada uno de los datasets con los hiperparámetros considerados en [3], descritos en la tabla 2.

En la figura 14 podemos observar una gráfica que muestra la evolución del fitness máximo obtenido por los individuos de la población en el transcurso del algoritmo CHC, utilizando la misma configuración de hiperparámetros en todos los datasets. En el eje horizontal se muestra la progresión del número de evaluaciones efectuadas y en el eje vertical el valor obtenido en la función de fitness del mejor individuo de la población.

Como se puede apreciar, la convergencia del algoritmo es bastante rápida estancándose en la mayoría de los casos antes de las 10000 evaluaciones, a pesar de ello, se puede observar como CHC realiza una muy buena exploración inicial del espacio de búsqueda, aumentando de manera exponencial el fitness máximo de la población. Esto podría ser indicador de que la población sostiene una diversidad satisfactoria en una buena parte de la ejecución del algoritmo gracias a los mecanismos de prevención de incesto y reinicialización introducidos en CHC. Por otro lado, cabe destacar que los valores más altos de fitness han sido obtenidos para los datasets con una dimensionalidad más alta (wisconsin, twonorm, pageblocks y ring) frente a los datasets con menor número de variables que han obtenido, en general, peores resultados (bupa, tae, haberman y phoneme).

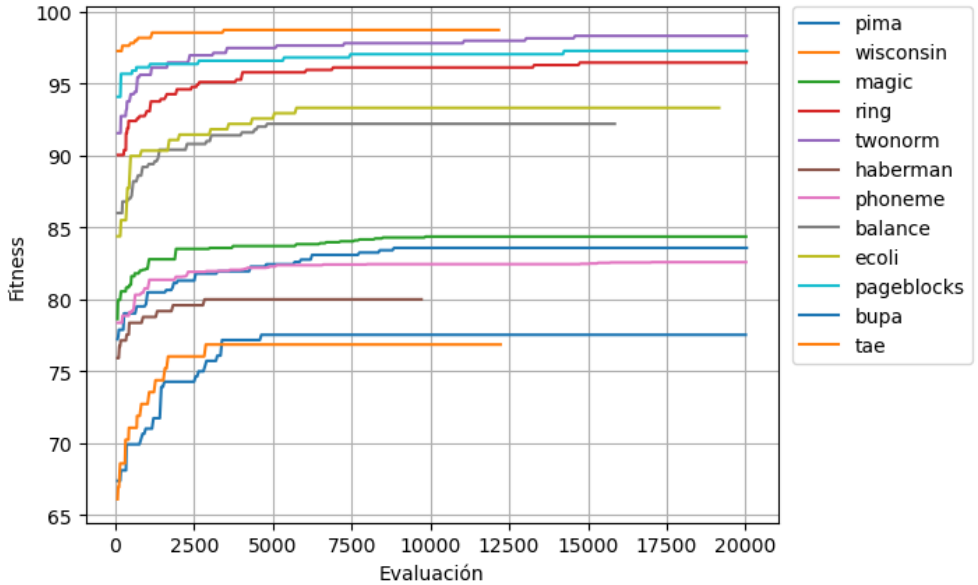

Figura 14: Evolución del fitness en CHC

En la tabla 3 se presentan los resultados obtenidos para el clasificador FARC-HD implementando el algoritmo CHC, en concreto, se muestran tres columnas de resultados, la primera columna hace referencia a la tasa de precisión alcanzada en los conjuntos de entrenamiento, la segunda columna hace referencia a la tasa de precisión alcanzada en los conjuntos de prueba y la última columna nos indica el tiempo consumido en segundos por el modelo.

En general los resultados alcanzados son bastante buenos, obteniendo una tasa de precisión media de un 88,64% en los conjuntos de entrenamiento, una tasa de precisión media de un 81,36% en los conjuntos de prueba y un tiempo medio de ejecución de 53,30 segundos, destacando los altos tiempos de ejecución requeridos por los datasets Magic y Phoneme. Además, se puede apreciar que en ninguno de los datasets hay una diferencia significativa en el rendimiento obtenido entre ambos conjuntos, esto podría indicar que el algorimo CHC no sufre de over-fitting y generaliza bien ante nuevos patrones.

Tabla 3: PRECISIÓN DEL MODELO FARC-HD IMPLEMENTANDO EL ALGORITMO CHC

Dataset	**Train**	**Test**	**Runtime**
Pima	83,82	74,09	60,05
Wisconsin	98,83	96,92	21,04
Magic	84,25	80,97	118,72
Ring	97,09	90,54	47,35
Twonorm	97,80	90,27	74,28
Haberman	81,21	74,49	8,16
Phoneme	83,28	81,51	189,84
Balance	92,16	85,76	48,09
Ecoli	92,19	82,14	27,70
Page-blocks	97,08	93,97	20,53
Bupa	79,42	66,09	13,67
Tae	76,49	59,61	10,18
Mean	88,64	81,36	53,30

3.3 FARC-HD IMPLEMENTANDO EL ALGORITMO GENÉTICO CLÁSICO

Para el estudio del algoritmo genético clásico se han realizado un total de 48 ejecuciones sobre el conjunto de datasets propuestos, alterando en cada ejecución la combinación de hiperparámetros utilizada por el algoritmo para poder estudiar la influencia de estos en el proceso de aprendizaje. En concreto, se han alterado la probabilidad de cruzamiento, la probabilidad de mutación, el método de selección utilizado y el método de reemplazamiento. En la tabla 4 se muestran los valores propuestos para cada hiperparámetro.

Tabla 4: HIPERPARÁMETROS UTILIZADOS EN EL ALGORITMO GENÉTICO CLÁSICO

Hiperparámetros
Probabilidad de cruzamiento = 0.8, 0.9
Probabilidad de mutación = 0.03, 0.01
Método de selección = Random, Ruleta, Torneo, Ranking
Método de reemplazamiento = ER, GR, BKPR

La gráfica de la figura 15 muestra la evolución del fitness en el proceso de aprendizaje del algoritmo genético clásico para la configuración de hiperparámetros que mejor tasa de precisión media ha obtenido en el conjunto de prueba.

Nótese que, a diferencia de la evolución en CHC, la convergencia del fitness en el algoritmo genético clásico, por lo general, es mucho más gradual y sostenida en el tiempo. Por otro lado, la tendencia que siguen los valores de la gráfica parece indicar que el algoritmo, en la mayoría de los casos y a excepción de los datasets tae y pageblocks que sí que parece converger a un óptimo en unas 2500 evaluaciones, no llega a converger del todo, haciéndonos pensar que este algoritmo requiere de un mayor número de iteraciones para poder alcanzar su mejor rendimiento.

Por último, también es destacable que pese a parecer no haber llegado a un punto de estancamiento, el algoritmo arroja resultados bastante competitivos con CHC manteniendo su mismo patrón de rendimiento, obteniendo en general, mejores resultados en los datasets de alta dimensionalidad.

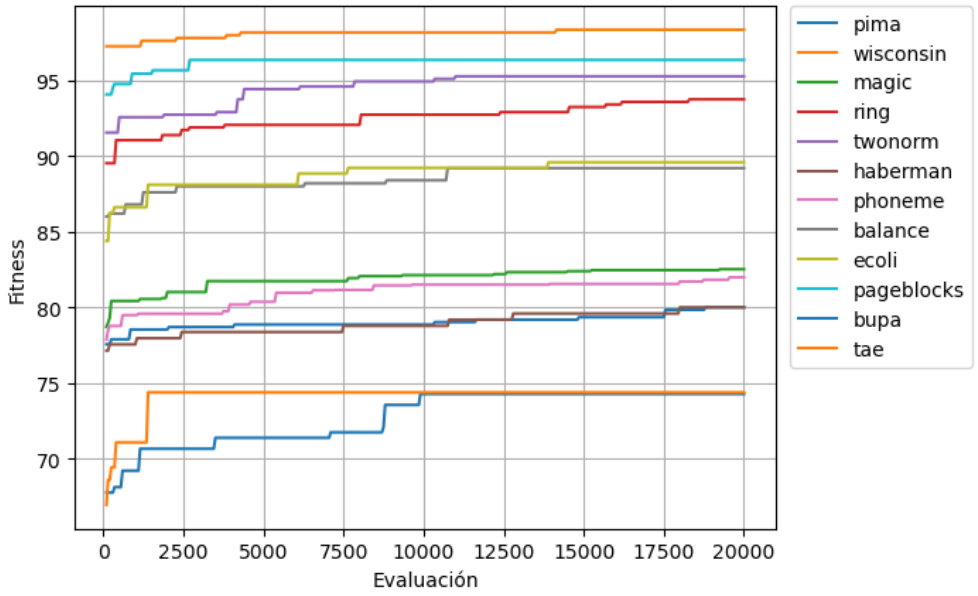

Figura 15: Evolución del fitness en el algoritmo genético clásico

En la tabla 5 se presentan los resultados obtenidos para el clasificador FARC-HD implementando el algoritmo genético clásico, resaltando en negrita los mejores. En concreto, se muestra la configuración de hiperparámetros y los resultados obtenidos en los conjuntos de entrenamiento, prueba y tiempo de ejecución de las cinco mejores ejecuciones del algoritmo.

Se puede observar como la mejor tasa de precisión media obtenida para los conjuntos de entrenamiento ha sido de un 86,32%, con solo un 2,32% de diferencia con el algoritmo CHC, y de un 80,82% para los conjuntos de prueba con una diferencia del 0,54%. Además, ninguna de las ejecuciones supera el tiempo medio tomado por el algoritmo CHC, aun que esto no supone una ventaja por si misma debido a que, como hemos mencionado anteriormente todo apunta a que el algoritmo genético clásico no termina de converger en 20000 evaluaciones y por tanto llegar a su punto de estancamiento requerirá de un mayor consumo de tiempo, haciéndolo en ese aspecto, menos eficiente que el algoritmo CHC. Por otro lado, cabe resaltar que el método de selección por ruleta no ha sido escogido en ninguna de las cinco ejecuciones.

Por último, no existe mucha diferencia entre los resultados obtenidos en los conjuntos de entrenamiento y prueba, por lo que al igual que en el algoritmo CHC, podemos decir que este algoritmo está generalizando de forma correcta y no sufre over-fitting.

Tabla 5: PRECISIÓN DEL MODELO FARC-HD IMPLEMENTANDO EL ALGORITMO GENÉTICO CLÁSICO

	Configuración 1 Pc=0.8, Pm=0.01, Torneo, ER			**Configuración 2** Pc=0.8, Pm=0.01, Ranking, ER			**Configuración 3** Pc=0.9, Pm=0.01, Ranking, BKPR			**Configuración 4** Pc=0.8, Pm=0.03, Random, GR			**Configuración 5** Pc=0.8, Pm=0.01, Random, GR		
Dataset	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**
Pima	80,34	75,00	**57,31**	80,14	74,60	57,93	79,98	75,51	58,69	79,92	**76,56**	59,25	**80,44**	74,73	60,11
Wisconsin	98,32	**97,07**	24,63	98,21	96,63	23,71	**98,43**	96,63	25,46	98,28	96,78	**23,61**	98,24	96,49	24,25
Magic	81,86	**79,86**	128,25	82,05	**79,86**	129,02	**82,14**	79,81	131,77	81,93	79,65	127,85	82,08	79,18	133,22
Ring	94,22	91,62	41,87	94,22	90,54	40,66	94,22	90,54	41,29	94,12	**91,76**	39,19	**94,66**	**91,76**	40,96
Twonorm	**95,17**	89,46	61,49	95,07	**90,27**	59,62	94,73	89,19	59,98	94,83	89,59	**55,91**	94,76	89,46	59,84
Haberman	**80,06**	72,19	10,85	79,82	73,51	**10,28**	**80,06**	73,17	11,02	79,90	**73,83**	10,80	**80,06**	72,53	10,72
Phoneme	**82,03**	80,79	188,03	81,45	80,51	188,96	81,62	81,01	184,04	81,09	80,87	**183,71**	81,81	**81,33**	185,57
Balance	90,12	86,56	54,35	89,96	86,40	53,90	89,80	85,12	51,46	89,76	85,44	50,96	**90,24**	**86,72**	52,39
Ecoli	88,69	79,18	24,85	**89,66**	80,08	25,19	88,99	79,77	24,05	88,99	**81,54**	23,94	89,29	79,77	25,23
Pageblocks	96,40	94,34	20,41	**96,49**	**94,52**	19,48	96,30	93,97	19,99	96,30	**94,52**	**19,01**	96,40	**94,88**	19,68
Bupa	74,35	**68,12**	12,63	74,71	66,38	12,73	74,28	**66,96**	12,83	73,26	63,19	12,57	**75,29**	64,06	12,82
Tae	**73,84**	55,63	11,35	72,68	56,28	12,23	72,02	**56,97**	11,57	72,35	53,63	11,62	72,52	56,30	**11,31**
Mean	86,28	**80,82**	53,00	86,20	80,80	52,81	86,05	80,72	52,68	85,89	80,61	**51,54**	**86,32**	80,60	53,01

3.4 FARC-HD IMPLEMENTANDO EL ALGORITMO DE EVOLUCIÓN DIFERENCIAL

Para el estudio del algoritmo de evolución diferencial se han realizado un total de 40 ejecuciones sobre el conjunto de datasets propuestos. Los hiperparámetros alterados para la prueba de este algoritmo han sido la probabilidad de cruzamiento, el modelo de mutación diferencial utilizado, el tipo de cruzamiento empleado y el hiperparámetro de escalado F. En la tabla 6 se muestran los valores propuestos para cada hiperparámetro.

Tabla 6: HIPERPARÁMETROS UTILIZADOS EN EL ALGORITMO DE

Hiperparámetros
Probabilidad de cruzamiento = 0.01, 0.9
Mutación diferencial = DE/rand/1, DE/best/1, DE/current-to-best/1, DE/rand/2, DE/best/2
Tipo de cruzamiento = Binomial, Exponencial
Parámetro de escalado F = 0.5, 0.9

La gráfica de la figura 16 muestra la evolución del fitness en el proceso de aprendizaje del algoritmo de evolución diferencial para la configuración de hiperparámetros que mejor tasa de precisión media ha obtenido en el conjunto de prueba.

Como se puede comprobar, pese a haber utilizado un valor alto para el hiperparámetro de escalado F de modo que se otorga mayor peso a los vectores diferencia, haciendo que el vector donante resultante se aleje del vector base lo cual propicia un comportamiento más exploratorio y de tener una población de un tamaño considerable (50 individuos) lo cual aumenta la diversidad introducida en el algoritmo, los resultados obtenidos en esta gráfica son bastante peculiares, prácticamente el algoritmo se estanca en muy pocas iteraciones, haciendo que el fitness máximo de la población mejore muy poco o no mejore en el transcurso del algoritmo.

Como sabemos, el algoritmo de evolución diferencial se caracteriza por un periodo inicial de exploración el cual depende principalmente de la diversidad en su población y un periodo de explotación de las regiones más prometedoras. A medida que el algoritmo empieza a converger, esta diversidad se va perdiendo dando como resultado individuos más parecidos entre sí y, como consecuencia, los vectores diferencia se van reduciendo generando individuos cada vez más cercanos a los vectores objetivo. Este hecho, sumado al reemplazo elitista que se produce en el

algoritmo de evolución diferencial puede hacer que la población se concentre rápidamente en los mejores óptimos encontrados, induciéndolo a una convergencia prematura.

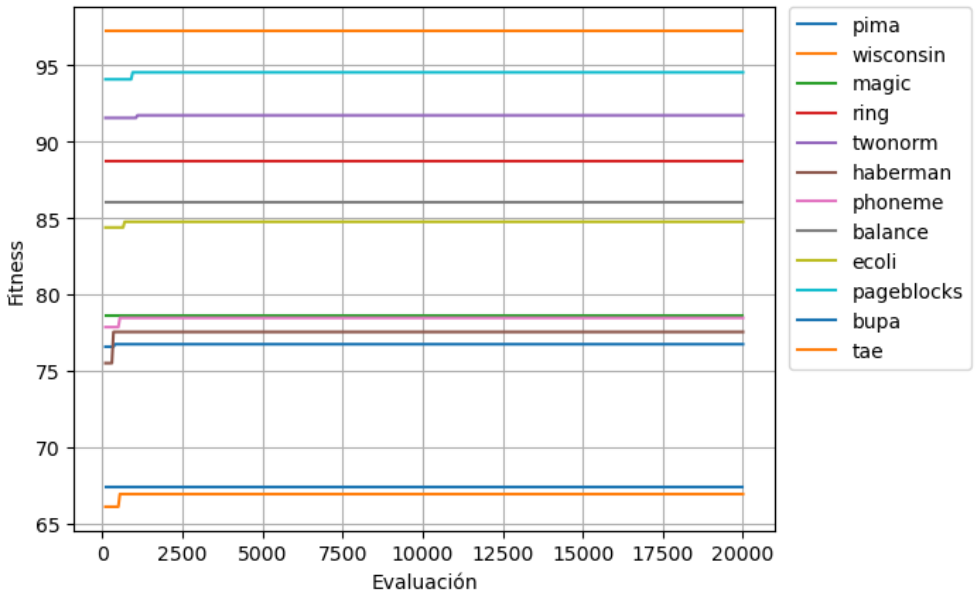

Figura 16: Evolución del fitness en el algoritmo DE

Como se puede observar en la tabla 7, a pesar de la rápida convergencia del algoritmo, este es capaz de obtener resultados bastante competitivos con CHC, llegando a una tasa de precisión media de un 82,56% en los conjuntos de entrenamiento con una diferencia de 6,08% en comparación con CHC y una tasa media de un 79,10% en los conjuntos de prueba con una diferencia del 2,26%. Por otro lado, todo parece indicar que los modelos de mutación diferencia DE/rand/ parecen trabajar mejor cuando la probabilidad de cruzamiento es alta, mientras que los modelo DE/best/ parecen hacerlo con probabilidades de cruzamiento bajas. Además, el hiperparámetro de escalado con un valor mayor y el cruce exponencial parecen obtener los mejores resultados.

Tabla 7: PRECISIÓN DEL MODELO FARC-HD IMPLEMENTANDO EL ALGORITMO DE

	Cr=0.9, DE/rand/2, exponencial, F=0.9			**Cr=0.01, DE/best/2, exponencial, F=0.9**			**Cr=0.9, DE/rand/1, exponencial, F=0.9**			**Cr=0.01, DE/best/1, binomial, F=0.9**			**Cr=0.01, DE/best/2, exponencial, F=0.5**		
Dataset	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**
Pima	76,95	72,78	53,79	76,99	73,82	51,79	76,76	73,43	59,13	76,76	73,82	71,12	**77,05**	**75,12**	**51,06**
Wisconsin	97,07	**97,07**	23,03	**97,22**	96,93	**22,57**	97,07	**97,07**	22,91	97,18	96,92	22,84	**97,22**	**97,07**	24,99
Magic	78,21	77,60	144,40	78,08	77,29	**124,87**	78,15	76,81	126,58	78,27	**78,34**	147,37	**78,36**	77,29	148,01
Ring	90,34	**89,73**	**46,77**	90,24	88,65	53,73	**90,57**	89,32	48,48	90,30	88,92	63,95	90,44	88,38	52,07
Twonorm	**92,50**	89,05	79,86	92,33	**89,19**	**79,14**	92,33	88,65	79,79	92,30	88,51	93,42	92,23	88,78	81,47
Haberman	**76,96**	**73,18**	9,79	76,63	72,20	9,54	76,63	72,53	9,84	76,88	72,21	9,83	76,71	71,55	**9,45**
Phoneme	78,52	**78,59**	219,65	78,56	77,89	206,82	**78,58**	78,15	**196,07**	78,54	78,03	244,95	78,51	78,00	209,24
Balance	85,32	82,88	**52,57**	85,68	83,68	56,59	85,56	**84,00**	58,07	**85,80**	83,52	63,99	85,48	83,04	58,00
Ecoli	84,08	77,38	32,72	84,30	**80,36**	31,19	85,27	78,58	**29,89**	**83,56**	77,09	37,82	**83,56**	78,28	32,37
Pageblocks	94,71	**93,61**	**18,77**	**94,89**	93,43	20,29	94,75	**93,61**	20,81	94,71	93,06	21,56	94,75	93,24	20,24
Bupa	67,61	**63,19**	13,49	67,54	62,32	13,26	**68,12**	61,74	**12,30**	67,46	62,61	15,13	67,97	**63,19**	13,68
Tae	65,39	54,34	9,60	66,23	53,66	9,47	**66,56**	54,97	**9,03**	65,56	**55,68**	11,32	66,23	54,32	10,15
Mean	82,30	**79,12**	58,70	82,39	79,12	56,61	**82,53**	79,07	**56,07**	82,28	79,06	66,94	82,38	79,02	59,23

3.5 FARC-HD IMPLEMENTANDO EL ALGORITMO PSO

Para el estudio del algoritmo PSO se han realizado un total de 36 ejecuciones sobre el conjunto de datasets propuestos. Los hiperparámetros alterados para la prueba de este algoritmo han sido la ratio de aprendizaje para la componente cognitiva φ_1, la ratio de aprendizaje para la componente social φ_2, el hiperparámetro de ajuste de inercia α y el modelo de PSO utilizado. En la tabla 8 se muestran los valores propuestos para cada hiperparámetro.

Tabla 8: HIPERPARÁMETROS UTILIZADOS EN EL ALGORITMO PSO

Hiperparámetros
Ratio de aprendizaje φ_1 = 0, 1, 2
Ratio de aprendizaje φ_2 = 0, 1, 2
Ajuste de inercia α = 0, 100
Modelo de PSO = Local, Global

Como se puede observar en la figura 17, la evolución del fitness en el proceso de aprendizaje del algoritmo PSO para la configuración de hiperparámetros que mejor tasa de precisión media ha obtenido en el conjunto de prueba, es más similar a la obtenida por el algoritmo CHC.

Además, podemos apreciar como en las primeras 5000 evaluaciones aproximadamente el algoritmo realiza una muy buena exploración del espacio de búsqueda, aumentando de forma muy significativa el fitness máximo alcanzado por la población. También podemos observar como tras esta etapa, el algoritmo continúa mejorando el fitness de una manera más pausada lo que podría indicar que se está haciendo una búsqueda más extensiva cerca de los óptimos encontrados. Es importante destacar que, a diferencia del algoritmo genético clásico, este algoritmo sí parece llegar en la mayoría de los casos a una buena convergencia para el número máximo de evaluaciones fijado y al igual que en el algoritmo CHC y el algoritmo genético clásico, también se sigue el mismo patrón de rendimiento, obteniendo mejores resultados para los datasets con mayor dimensionalidad a diferencia de los datasets con menor número de variables.

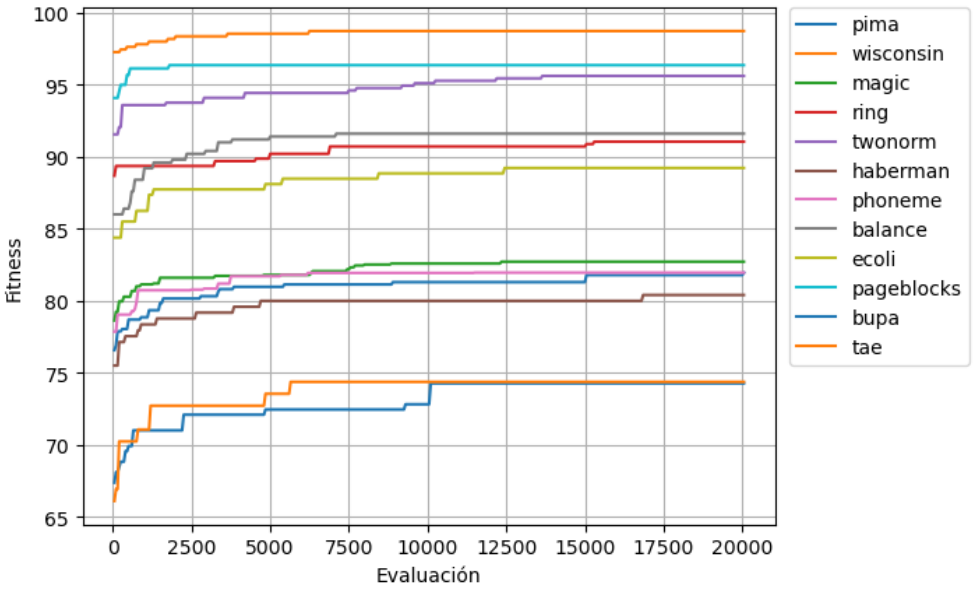

Figura 17: Evolución del fitness en el algoritmo PSO

En la tabla 9 se presentan los resultados obtenidos para el clasificador FARC-HD implementando el algoritmo PSO, resaltando en negrita los mejores. Como se puede observar se ha alcanzado a una tasa de precisión media de un 86,84% en los conjuntos de entrenamiento con una diferencia del 1,8% en comparación con CHC y una tasa media de un 80,87% en los conjuntos de prueba con una diferencia del 0,49%.

También podemos observar como el hiperparámetro de ajuste de inercia ha tomado el valor 100 en todas las ejecuciones, esto nos indica que el algoritmo ha hecho uso del parámetro de inercia ω para encontrar las mejores soluciones, señalando su importancia para encontrar un buen balance entre la exploración y la explotación del espacio de búsqueda. Por otro lado, todo parece indicar que la componente social ha sido la más beneficiosa para el algoritmo, y como se puede comprobar la ejecución que menos tiempo ha tomado es la que no utiliza la componente cognitiva. En general, los resultados obtenidos por el algoritmo PSO son bastante buenos, siendo el algoritmo que más se acerca al rendimiento conseguido por CHC.

Tabla 9: PRECISIÓN DEL MODELO FARC-HD IMPLEMENTANDO EL ALGORITMO PSO

	Local,φ_1=1, φ_2=2, α =100			**Local, φ_1=0, φ_2=2, α =100**			**Global, φ_1=1, φ_2=1, α =100**			**Local, φ_1=2, φ_2=1, α =100**			**Local, φ_1=1, φ_2=1, α =100**		
Dataset	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**
Pima	**81,64**	**76,30**	59,16	81,15	76,17	**54,86**	81,38	74,47	57,41	81,41	74,48	62,18	80,92	75,26	56,52
Wisconsin	98,68	96,19	24,96	98,57	**96,92**	**24,42**	**98,87**	96,49	25,18	98,57	96,34	25,58	98,57	97,22	25,11
Magic	82,45	80,07	119,54	82,28	78,70	**112,69**	82,47	**80,81**	117,29	82,54	80,60	114,37	**82,64**	80,02	115,36
Ring	93,11	89,73	41,42	93,07	89,46	**38,53**	**93,61**	89,05	40,70	93,41	89,32	42,71	94,43	**90,68**	42,95
Twonorm	95,10	90,00	66,93	94,22	89,32	**61,29**	**95,61**	88,92	64,51	**95,61**	89,46	70,98	95,10	**90,41**	71,92
Haberman	80,80	68,94	10,90	80,39	**72,54**	**10,43**	80,88	71,22	10,57	**81,13**	71,89	11,11	80,31	72,52	10,58
Phoneme	81,75	81,07	200,80	81,51	80,79	**192,24**	**82,37**	80,72	202,97	82,24	**81,31**	201,97	82,18	81,01	197,22
Balance	91,40	88,16	56,67	91,32	89,28	**51,97**	91,52	87,84	53,04	**91,68**	**89,60**	55,74	91,60	88,32	52,80
Ecoli	88,99	**80,38**	23,98	88,10	78,27	**23,73**	89,06	79,78	24,14	87,87	78,59	25,96	**89,58**	79,47	26,29
Pageblocks	96,58	94,52	20,29	96,21	94,34	19,06	**96,67**	94,34	19,67	96,49	**94,71**	**18,75**	96,40	93,61	19,23
Bupa	75,43	68,12	13,15	75,07	**68,41**	**12,24**	75,51	66,96	12,74	**75,94**	62,61	12,57	75,14	62,32	12,25
Tae	73,68	**56,95**	11,47	73,35	55,68	10,59	74,17	56,32	11,56	**74,34**	57,61	11,11	**74,34**	55,61	11,14
Mean	86,63	**80,87**	54,10	86,27	80,82	**51,00**	**86,84**	80,58	53,31	86,77	80,54	54,42	86,77	80,54	53,45

3.6 COMPARATIVA

En este último apartado se va a realizar una puesta en común de los mejores resultados obtenidos para cada una de las propuestas realizadas en este proyecto (véase tabla 11), además, se realizará la prueba no paramétrica de rangos con signo de Wilcoxon [37] para estudiar si la diferencia entre CHC y los algoritmos propuestos se debe al azar o podemos afirmar que existen diferencias estadísticamente significativas entre ellas.

Como se puede observar en la gráfica comparativa de la figura 18, CHC ha sido el algoritmo que mejores resultados a alcanzado, seguido del algoritmo PSO y el algoritmo genético clásico, el algoritmo que peores resultados ha obtenido ha sido el algoritmo de evolución diferencial. También es destacable que, aunque los resultados obtenidos en los conjuntos de entrenamiento para CHC son notablemente más altos que en el resto de las propuestas, esa diferencia no es tan clara en los conjuntos de prueba donde es mucho más escasa.

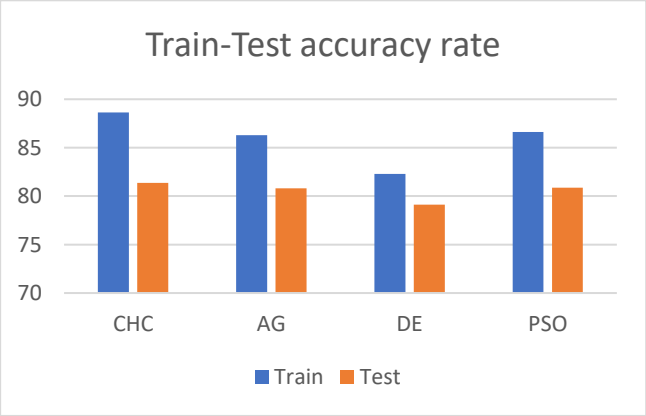

Figura 18: Comparativa de la tasa de precisión obtenida con los diferentes algoritmos

Por otro lado, como se puede observar en la tabla 10 el test de rangos con signo de Wilcoxon nos muestra como no se encuentran diferencias estadísticas entre CHC y los algoritmos PSO y genético clásico, en cambio, si podemos decir que el algoritmo DE es estadísticamente peor que CHC.

Tabla 10: Test de Wilcoxon para comparar CHC vs algoritmos propuestos (α=0.05)

Comparativa	R^+	R^-	p-valor
$FARCHD_{CHC}\ vs\ FARCHD_{AG}$	49	29	0.435
$FARCHD_{CHC}\ vs\ FARCHD_{DE}$	77	1	0.002
$FARCHD_{CHC}\ vs\ FARCHD_{PSO}$	48	30	0.477

Tabla 11: Comparativa resultados obtenidos en CHC vs propuestas

	CHC			AG			DE			PSO		
Dataset	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**	**Train**	**Test**	**Time**
Pima	**83,82**	74,09	60,05	80,34	75,00	57,31	76,95	72,78	53,79	81,64	**76,30**	59,16
Wisconsin	**98,83**	96,92	21,04	98,32	**97,07**	24,63	97,07	**97,07**	23,03	98,68	96,19	24,96
Magic	**84,25**	**80,97**	118,72	81,86	79,86	128,25	78,21	77,60	144,40	82,45	**80,07**	119,54
Ring	**97,09**	90,54	47,35	94,22	**91,62**	41,87	90,34	89,73	46,77	93,11	89,73	41,42
Twonorm	**97,80**	**90,27**	74,28	95,17	89,46	61,49	92,50	89,05	79,86	95,10	90,00	66,93
Haberman	**81,21**	**74,49**	8,16	80,06	72,19	10,85	76,96	73,18	9,79	80,80	68,94	10,90
Phoneme	**83,28**	**81,51**	189,84	82,03	80,79	188,03	78,52	78,59	219,65	81,75	81,07	200,80
Balance	**92,16**	85,76	48,09	90,12	86,56	54,35	85,32	82,88	52,57	91,40	**88,16**	56,67
Ecoli	**92,19**	**82,14**	27,70	88,69	79,18	24,85	84,08	77,38	32,72	88,99	80,38	23,98
Pageblocks	**97,08**	93,97	20,53	96,40	94,34	20,41	94,71	93,61	18,77	96,58	**94,52**	20,29
Bupa	**79,42**	66,09	13,67	74,35	**68,12**	12,63	67,61	63,19	13,49	75,43	**68,12**	13,15
Tae	**76,49**	59,61	10,18	73,84	55,63	11,35	65,39	54,34	9,60	73,68	**56,95**	11,47
Mean	**88,64**	**81,36**	53,30	86,28	80,82	**53,00**	82,30	79,12	58,70	86,63	80,87	54,10

4 CONCLUSIONES Y LINEAS FUTURAS

Tal y como se ha podido comprobar a lo largo de este proyecto, los sistemas basados en reglas difusas y especialmente el sistema de clasificación FARC-HD son modelos de clasificación robustos y compactos a la hora de trabajar con conjuntos de datos de alta dimensionalidad, además, al componerse de varias etapas bien diferenciadas, estos sistemas permiten una alta modificación.

Este proyecto en concreto a tratado de buscar algunas de las alternativas más viables para reemplazar el algoritmo evolutivo utilizado en la tercera etapa de estos sistemas y tras la experimentación realizada, hemos podido llegar a la conclusión de que pese a no haber podido superar el rendimiento obtenido por el algoritmo CHC, los resultados obtenidos por las propuestas realizadas han sido bastante satisfactorios, convirtiéndolas en un punto de partida muy firme para continuar investigando nuevas alternativas que puedan superarlo.

Uno de los mayores retos al que nos hemos enfrentado en este proyecto ha sido la tarea de encontrar la mejor configuración de hiperparámetros para cada uno de los algoritmos propuestos, por una parte, esto se ha debido a que cada problema al que nos enfrentamos tiene características diferentes y su mejor configuración de hiperparámetros puede que no sea extrapolable al resto de problemas y por otro lado, existe una limitación de tiempo que impide el desempeño de una prueba extensa para cada uno de estos hiperparámetros. Aun así, se han encontrado configuraciones para los algoritmos propuestos que arrojan resultados muy competitivos con CHC demostrando su potencial.

Por último, este proyecto abre las puertas a diversas vías de investigación en torno a introducir mejoras en los algoritmos propuestos para mejorar su funcionamiento. Concretamente una de las vías que podemos tomar es la implementación de mecanismos de autoadaptación, de forma que la configuración del algoritmo se ajuste automáticamente en el trascurso del proceso evolutivo, algunas de las alternativas más conocidas para la evolución diferencial, por ejemplo, son el algoritmo SaDE, el algoritmo APTS o el algoritmo JADE entre otras. Con respecto al algoritmo PSO algunas de las alternativas más llamativas son el modelo Predator-pray el cual parece ser eficaz frente al estancamiento de la nube de partículas en óptimos locales, el modelo constricted PSO que ejerce un mejor control de la etapa de explotación del algoritmo o el utilizar diferentes topologías de vecindad para la componente social del algoritmo.

5 BIBLIOGRAFÍA

[1] Hilbert, M., & López, P. (2011). The world's technological capacity to store, communicate, and compute information. science, 332(6025), 60-65.

[2] Ishibuchi, H., Nakashima, T., & Nii, M. (2004). Classification and modeling with linguistic information granules: Advanced approaches to linguistic Data Mining. Springer Science & Business Media.

[3] Alcalá-Fdez, J., Alcala, R., & Herrera, F. (2011). A fuzzy association rule-based classification model for high-dimensional problems with genetic rule selection and lateral tuning. IEEE Transactions on Fuzzy systems, 19(5), 857-872.

[4] NetBeans, A. (s. f.). Welcome to Apache NetBeans. Recuperado de https://netbeans.apache.org/

[5] Project Jupyter. (s. f.). Home. Recuperado de https://jupyter.org/

[6] ¿Qué es el aprendizaje automático? | Microsoft Azure. (s. f.). Recuperado de https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-machine-learning-platform

[7] ¿Qué es la inteligencia artificial (IA)? (s/f). Oracle.com. Recuperado de https://www.oracle.com/es/artificial-intelligence/what-is-ai/

[8] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297.

[9] Kingsford, C., & Salzberg, S. L. (2008). What are decision trees?. Nature biotechnology, 26(9), 1011-1013.

[10] Murphy, K. P. (2006). Naive bayes classifiers. University of British Columbia, 18(60), 1-8.

[11] Taunk, K., De, S., Verma, S., & Swetapadma, A. (2019). A brief review of nearest neighbor algorithm for learning and classification. In 2019 International Conference on Intelligent Computing and Control Systems (ICCS) (pp. 1255-1260). IEEE.

[12] Krogh, A. (2008). What are artificial neural networks?. Nature biotechnology, 26(2), 195-197.

[13] Sanz Delgado, J. A., Sesma Sara, M., & Bustince Sola, H. (2021). A fuzzy association rule-based classifier for imbalanced classification problems. Information Sciences, 577, 265-279.

[14] Lógica difusa. (2022). Wikipedia, La enciclopedia libre. Recuperado de https://es.wikipedia.org/w/index.php?title=L%C3%B3gica_difusa&oldid=143680445

[15] Lógica borrosa. (s/f). Unileon.es. Recuperado de http://glossarium.bitrum.unileon.es/Home/logica-borrosa

[16] Bustince, H. (s/f). Tema1-0-Basic Concepts, Asignatura de Ingeniería del Conocimiento, Universidad Pública de Navarra.

[17] Zadeh, L. A. (1965). Fuzzy sets. Information and control, 8(3), 338-353.

[18] Apartado 7.3: Teoría de conjuntos difusos y lógica difusa. (s/f). Uma.es. Recuperado de http://www.lcc.uma.es/~eva/aic/apuntes/fuzzy.pdf

[19] Del Jesus, M. J. (1999). Aprendizaje evolutivo de sistemas de clasificación basados en reglas difusas [Tesis de doctorado no publicada]. Universidad de Granada.

[20] Fernández, S., Velasco, J. R., & López-Carmona, M. A. Sistema basado en reglas difusas para el mapeo de ontologías. In *Congreso Español Sobre Tecnologías y Lógica Fuzzy (ESTYLF 2010). Huelva, España*.

[21] Sánchez, L., & Couso, I. Sistemas basados en reglas difusas con datos imprecisos: nuevos retos.

[22] Sanz, J. A. (s/f). Tema7-Reglas de Asociación, Asignatura de Minería de Datos, Universidad Pública de Navarra.

[23] Ishibuchi, H., & Yamamoto, T. (2005). Rule weight specification in fuzzy rule-based classification systems. IEEE transactions on fuzzy systems, 13(4), 428-435.

[24] Kavšek, B., & Lavrač, N. (2006). APRIORI-SD: Adapting association rule learning to subgroup discovery. Applied Artificial Intelligence, 20(7), 543-583.

[25] Janga Reddy, M., & Nagesh Kumar, D. (2020). Evolutionary algorithms, swarm intelligence methods, and their applications in water resources engineering: a state-of-the-art review. H2Open Journal, 3(1), 135-188.

[26] Jong, K. D. (2009). Evolutionary computation. Wiley Interdisciplinary Reviews: Computational Statistics, 1(1), 52-56.

[27] Kicinger, R., Arciszewski, T., & De Jong, K. (2005). Evolutionary computation and structural design: A survey of the state-of-the-art. Computers & structures, 83(23-24), 1943-1978.

[28] Liu, Y., & Passino, K. M. (2000). Swarm intelligence: Literature overview. Department of electrical engineering, the Ohio State University.

[29] Eshelman, L. J. (1991). The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In Foundations of genetic algorithms (Vol. 1, pp. 265-283). Elsevier.

[30] Fernandez, F., & Pereira, G. (s/f). Tema3-1-Introducción a los algoritmos genéticos, Asignatura de Computación, Universidad Pública de Navarra.

[31] Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11(4), 341-359.

[32] Gong, T., & Tuson, A. L. (2007). Differential evolution for binary encoding. In Soft Computing in Industrial Applications (pp. 251-262). Springer, Berlin, Heidelberg.

[33] Kennedy, J., & Eberhart, R. (1995, November). Particle swarm optimization. In Proceedings of ICNN'95-international conference on neural networks (Vol. 4, pp. 1942-1948). IEEE.

[34] Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization. Swarm intelligence, 1(1), 33-57.

[35] subprocess — Subprocess management. (s. f.). Python documentation. Recuperado de https://docs.python.org/3/library/subprocess.html

[36] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera. (2011). KEEL Data-Mining Software Tool: Data Set Repository,

Integration of Algorithms and Experimental Analysis Framework. Journal of Multiple-Valued Logic and Soft Computing 17(2-3), 255-287.

[37] Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in statistics* (pp. 196-202). Springer, New York, NY.