META HEURÍSTICA DE OPTIMIZACIÓN MONO-OBJETIVO CONTINUA DE ALTA DIMENSIONALIDAD PARA EL ENTRENAMIENTO DE UNA MÁQUINA DE APRENDIZAJE EXTREMO

David Fernando Sotelo Montero
Angie Daniela Velásquez Garzón

Director: PhD. Carlos Alberto Cobos Lozada
Codirectora: PhD. Martha Eliana Mendoza Becerra

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas
Línea de Investigación en Sistemas Inteligentes
Popayán, Marzo de 2018

Tabla de contenido

Resumen 10

Capítulo 1 11

1. INTRODUCCIÓN 11

1.1 PLANTEAMIENTO DEL PROBLEMA 11
1.2 APORTES DEL PROYECTO 12
1.3 OBJETIVOS 13
1.3.1 Objetivo general 13
1.3.2 Objetivos específicos 13
1.4 RESULTADOS OBTENIDOS 13
1.5 ORGANIZACIÓN DEL DOCUMENTO 14

Capítulo 2 15

2. CONTEXTO TEÓRICO Y ESTADO DEL ARTE 15

2.1 CONTEXTO TEÓRICO 15
2.1.1 Redes neuronales de propagación hacia delante 15
2.1.2 Algoritmo de propagación hacia atrás 17
2.1.3 Máquina de aprendizaje extremo 17
2.1.4 Algoritmos meta-heurísticos 21
2.1.5 DECC-G 21
2.1.6 MOS 22
2.1.7 IHDELS 23
2.1.8 JMetal 24
2.2 ESTADO DEL ARTE 25

Capítulo 3 29

3. PROPUESTA 29

3.1 INTRODUCCIÓN 29
3.2 REPRESENTACIÓN DE UN INDIVIDUO 29
3.3 RESTRICCIONES DE UN INDIVIDUO 29
3.5 CONJUNTOS DE DATOS 30
3.6 INTEGRACIÓN CON EL FRAMEWORK JMETAL 32
3.6.1 Definición del problema 32
3.6.2 Modelos de validación 33
3.7 ADAPTACIÓN DE DECC-G 35
3.7.1 Adaptación de evolución diferencial 37
3.7.2 Adaptación de SaNSDE 39
3.8 ADAPTACIÓN DE MUESTREO DE MÚLTIPLES DESCENDIENTES 41
3.8.1 Adaptación de Solis And Wets 44
3.8.2 Adaptación de MTS-LS1 46
3.9 ADAPTACION DE IHDELS 50
3.9.1 Adaptación de ascenso a la colina 52

3.9.2 Adaptación de SaDE 53

Capítulo 4 55

4. EXPERIMENTACIÓN 55

4.1 AFINAMIENTO DE PARÁMETROS 55

4.1.1 Problemas de clasificación 56

4.1.2 Problemas de regresión 56

4.1.3 Afinamiento de los parámetros de DECC-G 56

4.1.4 Afinamiento de los parámetros de MOS 62

4.1.5 Afinamiento de los parámetros de IHDELS 64

4.1.6 Afinamiento de los parámetros de M-ELM 69

4.2 RESULTADOS 70

4.2.1 Comparación con M-ELM y caminata aleatoria 70

4.2.2 Análisis estadístico 73

4.2.3 Análisis de los conjuntos de datos 75

4.2.4 Análisis de resultados 85

Capítulo 5 87

5. CONCLUSIONES Y TRABAJO FUTURO 87

5.1 CONCLUSIONES 87

5.2 TRABAJO FUTURO 88

Capítulo 6 90

6. BIBLIOGRAFIA 90

Lista de Ecuaciones

Ecuación 1 Definición del error en BP. 17
Ecuación 2 Función sigmoidal. 20
Ecuación 3 Matriz de salida de la capa oculta. 20
Ecuación 4 Pesos de salida. 20
Ecuación 5 Salida de la SLFN. 21
Ecuación 6 Calculo de la exactitud de la SLFN. 21
Ecuación 7 Cálculo de la exactitud en problemas de regresión. 30
Ecuación 8 Evaluación objetivo de una solución. 32
Ecuación 9 Exactitud en validación cruzada. 35
Ecuación 10 Estrategia de reproducción con el esquema DE/rand/1. 38
Ecuación 11 Aplicación de la tasa de cruce para un individuo en cada gen. 38
Ecuación 12 Reemplazo de un individuo. 38
Ecuación 13 Selección de la estrategia de reproducción. 39
Ecuación 14 Estrategia de reproducción con el esquema DE/current to best/1. 39
Ecuación 15 Actualización de la probabilidad p. 39
Ecuación 16 Selección del método de cálculo del nuevo factor de escala. 40
Ecuación 17 Actualización de la probabilidad fp. 40
Ecuación 18 Valor de la tasa de cruce en la siguiente generación. 40
Ecuación 19 Valor de CRm en la siguiente generación. 40
Ecuación 20 Valor de wk para cada valor de CRrec. 41
Ecuación 21 Función de participación técnicas de MOS [7]. 42
Ecuación 22 Técnicas con mejor calidad MOS [7]. 42
Ecuación 23 Mejor calidad técnicas de MOS [7]. 42
Ecuación 24 Cálculo del decremento de participación de una técnica MOS [7]. 42
Ecuación 25 Incremento de participación de las mejores técnicas de MOS [7]. 43
Ecuación 26 Generación del nuevo individuo por suma de la longitud del avance. 44
Ecuación 27 Generación del nuevo individuo por resta de la longitud del avance. 44
Ecuación 28 Actualización del vector bias por suma. 44
Ecuación 29 Actualización del vector bias por resta. 45
Ecuación 30 Tasa de mejora de una búsqueda loca enl IHDELS. 50

Ecuación 31 Valor del factor de escala para un individuo..... 54

Ecuación 32 Tasa de cruce para un individuo..... 54

Ecuación 33 Valor de CRm para la siguiente generación..... 54

Ecuación 34. Diferencia crítica para el test de Nemenyi..... 74

Lista de Figuras

Figura 1 Representación de una FNN. 15

Figura 2 Definición de una SLFN. 19

Figura 3 Arquitectura de JMetal para la definición de un algoritmo y un problema 25

Figura 4 Representación de un individuo 29

Figura 5 Diagrama definición del problema ELM en JMetal 33

Figura 6 Diagrama de tipos de problemas 34

Figura 7 Representación de los subcomponentes en DECC-G. 36

Figura 8 Representación del a población de DE en DECC-G. 37

Figura 9 Individuo a optimizar en DE. 37

Figura 10 Multiplicación del individuo con pesos 37

Figura 11 Implementación técnica de MOS. 43

Figura 12 Implementación de las búsquedas locales en IHDELS. 52

Lista de Tablas

Tabla 1. Resumen de resultados de una FNN según el número de capas ocultas. Adaptado de [13]..... 17

Tabla 2. División de los conjuntos de datos para problemas de clasificación..... 31

Tabla 3. División de los conjuntos de datos para problemas de regresión 32

Tabla 4. Variables definidas por defecto en DECC-G 57

Tabla 5. CA (2, 6, 5) 57

Tabla 6. Vocabulario para CA de DECC-G 58

Tabla 7. Ranking del afinamiento de parámetros parte 1. Problemas de clasificación en DECC-G..... 58

Tabla 8. Ranking del afinamiento de parámetros parte 2. Problemas de clasificación en DECC-G..... 59

Tabla 9. Ranking final de CV. Problemas de clasificación en DECC-G 60

Tabla 10. Ranking final de TT. Problemas de clasificación en DECC-G..... 60

Tabla 11. *Ranking del afinamiento de parámetros parte 1. Problemas de regresión en DECC-G*..... 60

Tabla 12. Ranking del afinamiento de parámetros parte 2. Problemas de regresión en DECC-G..... 61

Tabla 13. Ranking final de CV. Problemas de regresión en DECC-G..... 61

Tabla 14. *Ranking final de TT. Problemas de regresión en DECC-G*..... 62

Tabla 15. Parámetros a utilizar en DECC-G 62

Tabla 16. Vocabulario para CA de MOS..... 63

Tabla 17. CA (2, 4, 5) 63

Tabla 18. *Ranking del afinamiento de parámetros. Problemas de clasificación y regresión en MOS*..... 63

Tabla 19. Parámetros a utilizar en MOS..... 64

Tabla 20. *CA (2, 12, 5)* 64

Tabla 21. Vocabulario par CA de IHDELS..... 66

Tabla 22. Vocabulario aplicado al CA de IHDELS..... 66

Tabla 23. *Ranking final de CV. Problemas de clasificación en IHDELS*..... 68

Tabla 24. Ranking final de TT. Problemas de clasificación en IHDELS 68

Tabla 25. *Ranking final de CV. Problemas de regresión en IHDELS* 68

Tabla 26. Ranking final de TT. Problemas de regresión en IHDELS..... 69

Tabla 27. Parámetros a utilizar en IHDELS 69

Tabla 28 Parámetros a utilizar en M-ELM 70

Tabla 29. Comparación de entrenamiento-prueba en clasificación 71

Tabla 30. Comparación de validación cruzada en clasificación 71

Tabla 31. Comparación de entrenamiento-prueba en regresión 72

Tabla 32. *Comparación de validación cruzada en regresión* 72

Tabla 33 Resultados de la prueba de Wilcoxon en problemas de clasificación 73

Tabla 34. *Prueba de Friedman en problemas de clasificación* 74

Tabla 35. Resultados prueba de Nemenyi en problemas de clasificación. 75

Tabla 36. Prueba de Friedman en problemas de regresión 75

Tabla 37 Características de los conjuntos de datos usados para el análisis 75

Tabla 38 Prueba de Friedman en grupo 1 según el balance de los conjuntos de datos. .. 77

Tabla 39 Prueba de Friedman en grupo 2 según el balance de los conjuntos de datos ... 77

Tabla 40. Resultados prueba de Nemenyi grupo 1 (desbalanceados). 78

Tabla 41 Prueba de Wilcoxon en grupo 1 según el balance de los conjuntos de datos. ... 78

Tabla 42 Prueba de Wilcoxon en grupo 2 según el balance de los conjuntos de datos. ... 79

Tabla 43 Prueba de Friedman en grupo 1 según el número de instancias de los conjuntos de datos. 80

Tabla 44 Prueba de Friedman en grupo 2 según el número de instancias de los conjuntos de datos. 80

Tabla 45. Resultados prueba de Nemenyi grupo 1 (menores a 450 instancias). 81

Tabla 46. Resultados prueba de Nemenyi grupo 2 (mayores a 450 instancias). 81

Tabla 47 Prueba de Wilcoxon en grupo 1 según el número de instancias de los conjuntos de datos. 82

Tabla 48 Prueba de Wilcoxon en grupo 2 según el número de instancias de los conjuntos de datos. 82

Tabla 49 Prueba de Friedman en grupo 1 según el número de atributos de los conjuntos de datos. 83

Tabla 50 Prueba de Friedman en grupo 2 según el número de atributos de los conjuntos de datos. 83

Tabla 51. Resultados prueba de Nemenyi grupo 1 (menos de 20 atributos). 84

Tabla 52 Prueba de Wilcoxon en grupo 1 según el número de atributos de los conjuntos de datos. 84

Tabla 53 Prueba de Wilcoxon en grupo 2 según el número de atributos de los conjuntos de datos. 85

Resumen

Una máquina de aprendizaje extremo (ELM) realiza el entrenamiento de una Red Neuronal de una sola Capa que se propaga hacia adelante (SLFN) en menor tiempo que el algoritmo de propagación hacia atrás. Una ELM define los pesos de entrada y sesgos de una SLFN con valores aleatorios, lo cual ocasiona que su desempeño disminuya. El presente trabajo de investigación centra sus esfuerzos en definir cuál de tres (3) de los mejores algoritmos meta-heurísticos mono-objetivo especializados en problemas de alta dimensionalidad de la competencia 2015 IEEE CEC (Congress on Evolutionary Computation), a saber: Coevolución cooperativa basada en evolución diferencial (DECC-G), Muestreo múltiple de descendientes (MOS) e Hibridación iterativa de evolución diferencial (DE) con búsqueda local con reinicio (IHDELS); consigue mejores resultados en exactitud o tiempo de una ELM. La evaluación de los algoritmos se realizó en conjuntos de datos de regresión y clasificación reconocidos por la comunidad académica, utilizando como modelos de validación el método de retención y validación cruzada.

Los resultados arrojan a IHDELS como la mejor meta-heurística de optimización para el problema de la ELM entre los algoritmos evaluados, con un ranking de 3.5526 al aplicar las pruebas no paramétricas de Friedman con el método de retención en problemas de clasificación. Superando los resultados obtenidos por el algoritmo del estado del arte Memético-ELM (M-ELM) el cual obtuvo un ranking de 6.3289. Generando así nuevo conocimiento en el área de la optimización del entrenamiento de redes neuronales que utilizan la ELM.

Capítulo 1

1. INTRODUCCIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

Las redes de propagación hacia adelante (Feedforward Neural Networks, FNN) son un tipo de red neuronal que se caracterizan por conectar todas las neuronas de una capa únicamente con las neuronas de la capa siguiente, es decir, el procesamiento de la información se realiza de manera unidireccional, ofreciendo grandes ventajas como aprendizaje rápido, facilidad de implementación y mínima intervención humana [1]. Una FNN con una única capa intermedia se conoce formalmente como Single-Hidden Layer Feedforward Network (SLFN).

En el estado del arte se reportan algoritmos que permiten realizar el entrenamiento de las FNN, entre ellos un método denominado máquina de aprendizaje extremo (Extreme Learning Machine, ELM) [2], que se especializa en el entrenamiento de las SLFN's. ELM es un algoritmo eficiente que a diferencia del algoritmo de propagación hacia atrás (backpropagation, BP) evita el proceso iterativo (diferentes épocas de entrenamiento) en el proceso de aprendizaje [3], el cual consume mucho tiempo en problemas con una cantidad significativa de datos [4]. ELM ha sido usado satisfactoriamente en diversas áreas del conocimiento como la ingeniería biomédica, la visión de computadoras, los sistemas de identificación, entre otras [5].

La teoría de ELM indica que los pesos de entrada (pesos entre la capa de entrada y la capa oculta) y los sesgos de la capa oculta se pueden asignar aleatoriamente mientras que los pesos de salida (pesos entre la capa oculta y la capa de salida) se deben asignar analíticamente por medio de la pseudoinversa de Moore-Penrose [4] solo cuando la función de activación de las neuronas de salida es lineal [6] y la función de activación de las neuronas de la capa oculta es continua e infinitamente diferenciable [5].

Resultados experimentales muestran que definir los pesos de entrada y los sesgos de la capa oculta en forma aleatoria afecta negativamente el desempeño de una SLFN, tendiendo a obtener menores valores de exactitud en problemas de clasificación, mayores valores de error en problemas de regresión y a requerir más neuronas en la capa oculta que al utilizar otros algoritmos de aprendizaje [6]. Por lo anterior, en [4] se usó un algoritmo memético (algoritmo genético con optimización local) que permite definir los pesos de entrada de una mejor forma y obtiene mejores resultados que cuando se definen aleatoriamente.

Por otro lado, existen algoritmos de optimización especializados en problemas con funciones multimodales continuas de alta dimensionalidad que no han sido utilizados para resolver el problema de la definición de los pesos de entrada en una

ELM, pero que dadas sus características podrían definir con más exactitud sus valores y obtener mejores resultados que los reportados en el estado del arte.

Entre estos algoritmos se encuentran MOS [7], DECC-G [8] e IHDELS [9] que fueron los algoritmos que presentaron mejores resultados en la competencia de optimización continua a gran escala en el 2015 IEEE CEC (Congress on Evolutionary Computation), donde se tienen en cuenta problemas continuos de 1000 y más dimensiones y funciones unimodales, multimodales, separables y no separables. Algunos de estos algoritmos además han sido recientemente utilizados en el programa de ingeniería de sistemas de la Universidad del Cauca, para la calibración de modelos de tráfico vehicular obteniendo excelentes resultados [10].

Considerando que el algoritmo propuesto en [4] mejora ostensiblemente los resultados de una SLFN entrenada con ELM, pero que es un algoritmo sencillo que no ha sido concebido para trabajar en alta dimensionalidad, en este proyecto se consideró que usar los algoritmos meta heurísticos MOS, DECC-G e IHDELS especializados en problemas de alta dimensionalidad y que han sido probados en diferentes paisajes de las funciones de optimización se podrían obtener resultados de entrenamiento superiores a los reportados en el estado del arte, y así obtener modelos de clasificación y regresión más exactos en la predicción de cada una de las áreas de aplicación.

Con lo anterior, en este proyecto se definió la siguiente pregunta de investigación: ¿En qué medida el uso de los algoritmos meta-heurísticos diseñados para problemas continuos de alta dimensionalidad MOS, DECC-G e IHDELS en el proceso de definición de los pesos de entrada en una SLFN entrenada con ELM permite obtener mejores resultados en problemas de clasificación o regresión o reducir el tiempo de aprendizaje en comparación con los métodos del estado del arte?

1.2 APORTES DEL PROYECTO

Desde un punto de vista investigativo, las contribuciones de este proyecto están enfocadas en generar nuevo conocimiento en el área de la optimización del entrenamiento de redes neuronales SLFN que usan ELM, utilizando la meta-heurística IHDELS especializada en alta dimensionalidad que arrojó los mejores resultados en comparación con las meta-heurísticas MOS, DECC-G y el algoritmo M-ELM. Este aporte se hace teniendo en cuenta que no se encontró ningún reporte de algoritmos de optimización para alta dimensionalidad enfocados al entrenamiento de redes neuronales mediante ELM, en las bases de datos de IEEE, ScienceDirect y Springer Link; y que el No-Free Lunch Theorem [11] establece que dado un tipo de problema de optimización específico, no se puede saber cuál es la mejor meta-heurística de optimización para abordarlo, estas deben ser evaluadas y comparadas experimentalmente.

Desde la perspectiva de innovación, en el proyecto se implementaron las meta-heurísticas de optimización para alta dimensionalidad MOS, DECC-G e IHDELS en

un framework que permita el entrenamiento de SLFN que usan ELM. Este producto queda a disposición de la comunidad académica, científica e industrial, con lo que se espera que se pueda usar en diversos problemas de optimización de la vida real.

1.3 OBJETIVOS

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

1.3.1 Objetivo general

Definir la mejor meta heurística de optimización mono-objetivo diseñada para resolver problemas continuos de alta dimensionalidad (MOS, DECC-G e IHDELS) que logra los mejores resultados en exactitud y/o tiempo en el entrenamiento de una máquina de aprendizaje extremo.

1.3.2 Objetivos específicos

- Definir la línea base de comparación del trabajo investigativo, contemplando los algoritmos con mejores resultados en el entrenamiento de una ELM reportados en el estado del arte, su implementación, las métricas y problemas de clasificación y regresión con los que la comunidad académica realiza la comparación.

- Adaptar las meta-heurísticas seleccionadas (MOS, DECC-G e IHDELS) al proceso de entrenamiento de una máquina de aprendizaje extremo, implementarlas y probarlas en problemas de clasificación y regresión.

- Evaluar el desempeño del entrenamiento de una ELM basado en los tres algoritmos meta-heurísticos y realizar un análisis comparativo de los resultados obtenidos frente a la línea base soportando dicho análisis con las pruebas estadísticas no paramétricas de Wilcoxon y Friedman.

1.4 RESULTADOS OBTENIDOS

A continuación, se presenta un resumen de los resultados más destacados que se obtuvieron en este trabajo de grado:

- **Monografía de trabajo de grado:** Se refiere al presente documento el cual presenta el estado del arte del problema y realiza una descripción de los algoritmos utilizados. Plantea la adaptación realizada al problema como un problema de optimización, presenta la integración de los algoritmos implementados y el problema de entrenamiento de redes neuronales con ELM al framework JMetal. Por último, expone los resultados obtenidos de la implementación de los algoritmos y su comparación con los algoritmos encontrados en el estado del arte.
- **Aplicación software:** Módulo en el framework JMetal de Java para el entrenamiento de una SLFN usando una máquina de aprendizaje extremo y los

problemas implementados, así mismo se incorporó al módulo de algoritmos en JMetal los algoritmos IHDELS, MOS y DECC-G, los cuales fueron implementados con el objetivo de poder optimizar un sin número de problemas siempre y cuando se encuentren definidos bajo ciertos parámetros.

- **Artículo**: un artículo de la presente monografía que se espera publicar en evento internacional. Ver **ANEXO II**.

1.5 ORGANIZACIÓN DEL DOCUMENTO

A continuación, se introducen los temas que se tratan en cada uno de los capítulos del resto del documento.

Capítulo 2: Contexto teórico y estado del arte. En este capítulo se presentan los conceptos necesarios para la comprensión del entrenamiento de una SLFN mediante el algoritmo ELM y algunos algoritmos sobresalientes en la optimización del entrenamiento usando ELM.

Capítulo 3: Propuesta. Este capítulo explica la representación del individuo y cómo fueron utilizados los modelos de validación dentro de la optimización. Además, las modificaciones realizadas y la integración de los algoritmos DECC-G, IHDELS y MOS al framework JMetal.

Capítulo 4: Experimentación. Este capítulo muestra las métricas utilizadas en el afinamiento de parámetros de las meta-heurísticas y las pruebas realizadas para seleccionar el mejor conjunto de valores para su ejecución. También, los resultados obtenidos en la ejecución con los conjuntos de datos de regresión y de clasificación en los dos modelos de validación y un análisis estadístico usando las pruebas no paramétricas de Wilcoxon y Friedman.

Capítulo 5: Conclusiones y trabajos futuros. En este capítulo se presentan las conclusiones obtenidas a partir de los resultados de los experimentos. Además, los trabajos futuros en la línea de investigación.

Por último, se presentan enumeradas todas las referencias bibliográficas utilizadas en el desarrollo del presente trabajo de grado.

Capítulo 2

2. CONTEXTO TEÓRICO Y ESTADO DEL ARTE

2.1 CONTEXTO TEÓRICO

La presente sección del documento centra su atención en exhibir los conceptos más significativos relacionados con el presente proyecto de investigación, con la intención de proveer claridad en la temática aquí abordada; para mayor detalle de estos se recomienda recurrir a las citas bibliográficas presentadas a través del capítulo.

2.1.1 Redes neuronales de propagación hacia delante

Las redes neuronales de propagación hacia delante (Feedforward Neural Networks, FNN) son un tipo de redes neuronales artificiales que topológicamente no tienen caminos cerrados [12], es decir, no existen ciclos en sus conexiones y consisten de un número de neuronas simples como unidades de procesamiento organizadas en capas [13]. Una FNN se caracteriza por tener una capa de entrada, una o varias capas ocultas y una capa de salida. Las neuronas en la capa de entrada no tienen conexiones hacia ellas, las neuronas en la capa de salida no tiene conexiones salientes de ellas [12], como se muestra en la ***Figura 1***. La información se mueve en una sola dirección desde las neuronas de la capa de entrada, a través de las neuronas de las capas ocultas (si existen) hasta las neuronas de la capa de salida, las neuronas están únicamente conectadas hacia delante, no existen conexiones hacia atrás, por consiguiente no existe retroalimentación entre las capas [13].

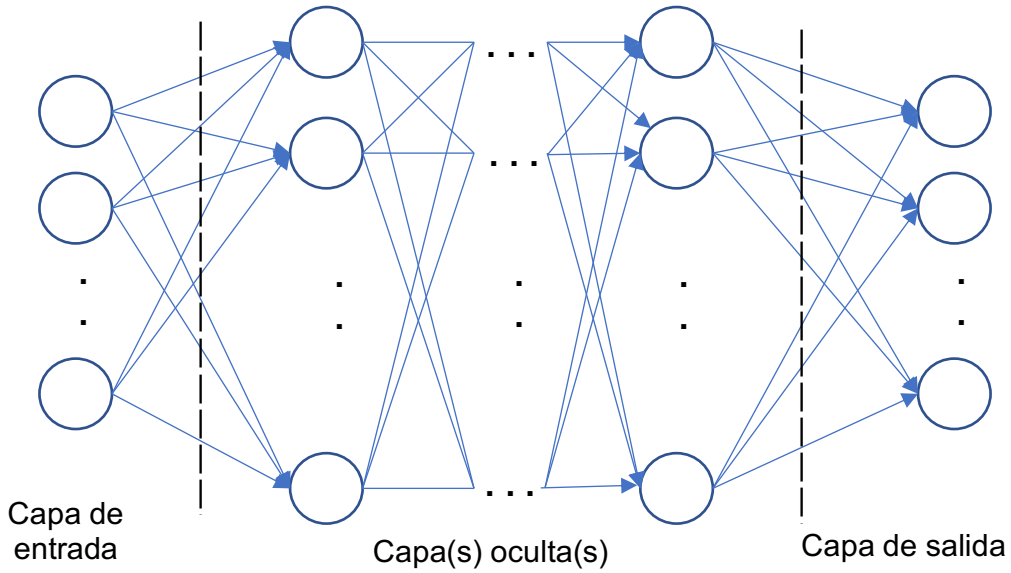

Figura 1 *Representación de una FNN.*

Una FNN es un algoritmo de clasificación o regresión inspirado en la biología, todas las neuronas de una capa están conectadas con todas las neuronas de la capa

contigua, estas conexiones no son iguales, cada conexión puede tener un peso diferente; estos pesos codifican el conocimiento de la red neuronal [13].

2.1.1.1 Estructura de una FNN

Una FNN puede construirse de diferentes maneras, generalmente el usuario decide la cantidad de neuronas en la capa de entrada (basado en los datos de entrada del problema a resolver), la cantidad de neuronas en la capa de salida (basado en el número de clases del problema de clasificación o una si es un problema de regresión), el número de capas ocultas y la cantidad de neuronas en cada una de las capas ocultas. Existen algunas reglas para tomar adecuadamente estas decisiones [13].

- **Capa de entrada** es el conducto por el cual se le presenta un patrón a la red neuronal, esta representa las condiciones para la cuales se lleva a cabo el entrenamiento de la red neuronal. Cada neurona de la capa de entrada representa una variable independiente que tiene una influencia sobre la salida de la red neuronal [13].
- **Capa de salida** presenta un patrón de salida el cual está directamente relacionado con la capa de entrada. El número de neuronas en la capa de salida deberá estar relacionado con el tipo de trabajo de la red neuronal. Si la red neuronal va a realizar una clasificación en grupos, es recomendable tener una neurona en la capa de salida por cada grupo a clasificar. Si la red neuronal va a realizar la reducción del ruido en una señal entonces es probable que el número de neuronas en la capa de entrada coincida con el número de neurona en la capa de salida [13].
- Existen dos decisiones importantes a tomar alrededor de las **capas ocultas**: 1) el número de capas ocultas y 2) la cantidad de neuronas en cada capa oculta. Una FNN con dos capas ocultas puede representar funciones con casi cualquier forma, sin embargo, es difícil encontrar problemas que requieran dos capas ocultas. Más aún para muchos problemas prácticos no existe razón alguna para utilizar más de una capa oculta [13]. En la ***Tabla 1*** se presenta un resumen de las diferencias que existen de acuerdo con el número de capas ocultas.
  Hay que mencionar además que el número de neuronas en las capas ocultas es una decisión de gran importancia, la clave para la eficacia de una red multicapa es que las neuronas en las capas ocultas aprendan a representar las variables de entrada de una manera dependiente a la tarea [14], dado que estas capas tienen una tremenda influencia en la salida de la red; tener pocas neuronas en las capas ocultas puede ocasionar problemas en la detección de señales en datos de entrenamiento complejos, por otro lado tener muchas neuronas en las capas ocultas incrementa el tiempo de entrenamiento y puede llevar a un sobreajuste (overfitting), que ocurre cuando la red tiene una capacidad de procesamiento tan alta que los datos de entrenamiento no son suficientes para entrenar todas las neuronas de la capa oculta [13].

Tabla 1. *Resumen de resultados de una FNN según el número de capas ocultas. Adaptado de [13]*

Número de capas ocultas	**Resultados**
0	Representa funciones linealmente separables o decisiones
1	Aproxima arbitrariamente cualquier función que contenga una asignación continua de un espacio finito a otro.

2.1.1.2 Aplicaciones de las FNN

Las FNN tienen grandes aplicaciones en problemas del mundo real, siendo exitosas en diferentes industrias gracias a que destacan por la identificación de patrones o tendencias en los datos lo cual las hace adecuadas para predicciones o pronósticos como: predicción de ventas, control de procesos, investigación de clientes, gestión de riesgos y marketing de objetivos. Las FNN también son un área de investigación llamativa en la medicina, se espera que tengan una amplia aplicación en sistema biomédicos en los próximos años [13].

2.1.2 Algoritmo de propagación hacia atrás

El algoritmo de propagación hacia atrás (Backpropagation, BP) es uno de los métodos de entrenamiento de redes neuronales con más popularidad, gracias a su sencillez, la idea principal radica en definir una función de error y utilizar el descenso del gradiente para encontrar el conjunto de pesos que optimiza una tarea específica [14].

El algoritmo BP es el más comúnmente implementado como procedimiento de entrenamiento de las FNN, su objetivo es proveer un mecanismo que permita actualizar los pesos de la red neuronal con base en la minimización del error, usualmente se utiliza el descenso del gradiente. El error es definido como lo expresa la ***Ecuación 1*** [13].

$$E(\vec{\omega}) = \frac{1}{2}\sum_{d\epsilon D}(t_d - o_d)^2$$

Ecuación 1 *Definición del error en BP.*

Donde d es un registro de los datos de entrenamiento D, t_d es la salida esperada y o_d es la salida de la red neuronal, finalmente $\vec{\omega}$ es el vector de pesos. El periodo de aprendizaje depende del tamaño de la FNN, los datos de entrenamiento, el número de patrones a aprender, el número de épocas y el mínimo error tolerable [13].

2.1.3 Máquina de aprendizaje extremo

Una máquina de aprendizaje extremo (Extreme Learning Machine, ELM) es un algoritmo de entrenamiento utilizada solamente para las FNN con una única capa

intermedia conocidas como SLFN (Single-Hidden Layer Feedforward Network). En una SLFN, los pesos entre las neuronas de entrada, el número de neuronas en la capa oculta y los sesgos de las neuronas de la capa oculta se inician aleatoriamente (independientemente de los datos de entrenamiento) y luego el valor de los pesos de la capa oculta a la capa de salida se calculan analíticamente basado en la seudo-inversa de Moorse-Penrose [5]. La principal ventaja de entrenar un SLFN con ELM es la velocidad del aprendizaje, la cual evita el proceso iterativo que se da con el backpropagation [5].

2.1.3.1 Funcionamiento de una ELM

El modelo de entrenamiento basado en ELM se define a continuación:

- **Definir datos de entrenamiento**

Definir los datos de entrada X y los datos de salida Y de la red neuronal, para así mismo establecer la cantidad de neuronas en la capa de entrada y la capa de salida, correspondientemente.

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1k} \\ x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mk} \end{bmatrix} = [x_1 \quad x_2 \quad \cdots \quad x_k], x_j = \begin{bmatrix} x_{1j} \\ x2_j \\ \vdots \\ x_{mj} \end{bmatrix}$$

Donde k es la cantidad de registros de entrenamiento, mientras que m corresponde a la cantidad de atributos de los datos de entrenamiento; es decir cada columna x_j representa un registro de los datos de entrenamiento.

$$Y = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1k} \\ y_{21} & y_{22} & \cdots & y_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ y_{l1} & y_{l2} & \cdots & y_{lk} \end{bmatrix}, \forall\, y_{ij} \in \{1, -1\} \wedge \forall\, y_{\cdot j} \left\| y_{\cdot j} = 1 \right\| = 1$$

Donde l corresponde al total de clasificaciones posibles de los datos de entrenamiento, es decir cada columna representa la salida de un registro de los datos de entrenamiento, por otro lado, cada fila representa una clasificación, el valor 1 se ubica exactamente una vez por cada columna y se asigna en la fila a la cual corresponda la clasificación de dicho registro en los datos de entrenamiento, el resto de los valores en una columna corresponderán al valor de -1. Para problemas de regresión Y es un arreglo donde cada columna representa el valor esperado de salida.

Una vez definidos los datos de entrenamiento se puede definir la SLFN que se utilizará como clasificador, la cantidad de neuronas en la capa de entrada y la capa de salida como se observa en la ***Figura 2***.

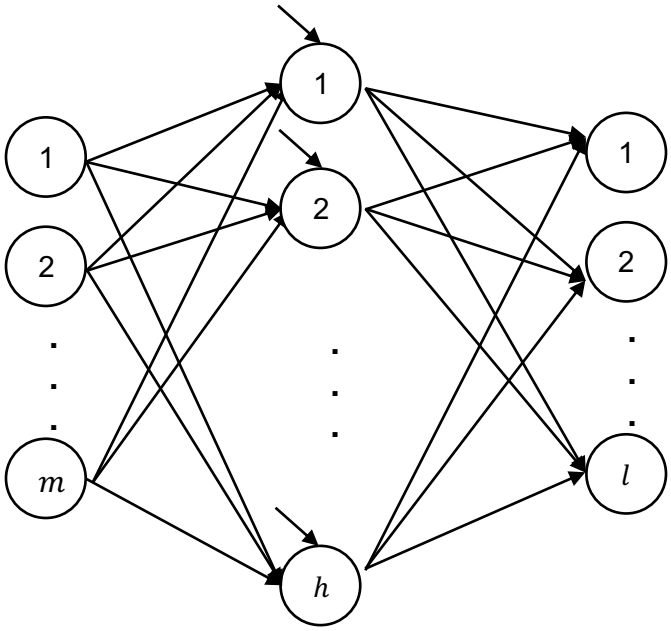

Figura 2 *Definición de una SLFN.*

La cantidad de neuronas en la capa de entrada coincide con la cantidad de atributos de los datos de entrada e igualmente la cantidad de neuronas en la capa de salida corresponde con la cantidad total de clasificaciones, por otra parte la cantidad de neuronas en la capa oculta es un valor que se toma de manera arbitraria, existen algoritmos como Self Adaptative ELM [15], que busca la cantidad adecuada de neuronas en la capa oculta para construir redes neuronales óptimas. Toma valores aleatorios en los parámetros de entrada y explora cuál es la cantidad de neuronas en la capa oculta que consigue que el desempeño de la red neuronal se optimice.

- **Definir pesos de entrada y sesgos**

Los pesos de entrada son definidos con valores aleatorios, representados en la matriz ω donde ω_{ij} es el peso entre la neurona i de la capa oculta y la neurona j de la capa de entrada donde $\forall\, i = \{1,2,\dots,h\} \wedge j = \{1,2,\dots,m\}$.

$$\omega = \begin{bmatrix} \omega_{11} & \omega_{12} & \cdots & \omega_{1m} \\ \omega_{21} & \omega_{22} & \cdots & \omega_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{h1} & \omega_{h2} & \cdots & \omega_{hm} \end{bmatrix} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_h \end{bmatrix}, \omega_i = \begin{bmatrix} \omega_{i1} & \omega_{i2} & \dots & \omega_{im} \end{bmatrix}$$

El vector de sesgos de la capa oculta se define como b, igualmente con valores aleatorios, donde cada fila hace referencia al sesgo de la i-esima neurona de la capa oculta.

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_h \end{bmatrix}$$

- **Definir la función de activación de la capa oculta**

Se define a $g(x)$ como la función de activación de la capa oculta, donde $x \in \mathbb{R}$. El presente trabajo utilizó como función de activación a la sigmoidal ver ***Ecuación 2***.

$$g(x) = \frac{1}{1 + \exp(-x)}$$

Ecuación 2 *Función sigmoidal.*

- **Calcular la matriz de salida de la capa oculta**

La matriz de salida de la capa oculta H se calcula con base a la cantidad de datos de entrenamiento y los pesos de entrada. En H, cada columna representa la multiplicación de los pesos de entrada de la i-esima neurona de la capa oculta con las neuronas de la capa de entrada más el sesgo de la correspondiente neurona oculta, ver ***Ecuación 3***.

$$H = \begin{bmatrix} g(\omega_1 x_1 + b_1) & g(\omega_2 x_1 + b_2) & \dots & g(\omega_h x_1 + b_h) \\ g(\omega_1 x_2 + b_1) & g(\omega_2 x_2 + b_2) & \dots & g(\omega_h x_2 + b_h) \\ \vdots & \vdots & \ddots & \vdots \\ g(\omega_1 x_k + b_1) & g(\omega_2 x_k + b_2) & \dots & g(\omega_h x_k + b_h) \end{bmatrix}$$

Ecuación 3 *Matriz de salida de la capa oculta.*

- **Calcular los pesos de salida**

Los pesos de salida β (pesos entre la capa oculta y la capa de salida) se calculan como se especifica en la ***Ecuación 4***, donde H^{+} es la inversa generalizada de Moore-Penrose de H y Y^{T} es la transpuesta de Y, donde β_{ik} representa el peso entre la neurona i de la capa oculta y la neurona k de la capa de salida $\forall\ i = \{1,2,\dots,h\} \wedge k = \{1,2,\dots,l\}$.

$$\beta = H^{+}Y^{T} == \begin{bmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1l} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{h1} & \beta_{h2} & \cdots & \beta_{hl} \end{bmatrix}$$

Ecuación 4 *Pesos de salida.*

- **Calcular la salida de la SLFN**

La salida de la red neuronal está denotada por T usando la ***Ecuación 5*** donde cada columna j representa la salida de un registro de entrenamiento y cada fila corresponde al total de clasificaciones posibles de los datos de entrenamiento. Por cada columna se identifica en cual fila se encuentra el mayor valor y dicha fila será la clasificación del j-esimo registro de entrenamiento.

$$T = (H\beta)^T = \begin{bmatrix} t_{11} & t_{21} & \dots & t_{l1} \\ t_{21} & t_{22} & \dots & t_{l2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{1k} & t_{2k} & \dots & t_{lk} \end{bmatrix}$$

Ecuación 5 *Salida de la SLFN.*

- **Calcular la exactitud de salida de la SLFN**

Para conocer la exactitud de la SLFN en problemas de clasificación, T y Y son contrastados, lo que implica verificar por cada columna que el mayor valor corresponda a la misma fila tanto en T como en Y, es decir que la clasificación de cada registro de entrenamiento (el cual está descrito en Y) sea el mismo que predijo la red neuronal. Cada acierto es contado y al finalizar se calcula el promedio entre aciertos sobre el total de registros de entrenamiento como se especifica en la ***Ecuación 6***.

$$exactitud = \frac{aciertos}{k}$$

Ecuación 6 *Calculo de la exactitud de la SLFN.*

Para calcular la exactitud en problemas de regresión, se comparan T y Y, se calcula el error cuadrático medio entre la salida esperada y la salida de la red neuronal.

2.1.4 Algoritmos meta-heurísticos

Los algoritmos meta-heurísticos son métodos para encontrar soluciones óptimas con un costo computacional razonable, para encontrar soluciones óptimas a un problema complejo, es decir, en un problema donde el espacio de búsqueda crece exponencialmente, el objetivo es encontrar una solución viable en un periodo de tiempo aceptable [16], pero no existe garantía de que la mejor solución (solución óptima global) sea encontrada [17].

Una meta-heurística se basa en dos conceptos importantes que son la diversificación y la intensificación. El proceso de diversificación se encarga de generar varias soluciones explorando el espacio de búsqueda global y seleccionando la mejor solución o varias de las mejores soluciones encontradas, por otro lado la intensificación se especializa en la búsqueda de soluciones candidatas en una región local aprovechándose del hecho que se ha encontrado una buena solución en dicha región [18].

Los algoritmos meta-heurísticos pueden ser clasificados de diversas maneras, entre ellas: basados en poblaciones, los cuales utilizan varios individuos durante su ejecución o basados en trayectorias, donde se utiliza un único individuo que se mueve sobre el espacio de búsqueda.

2.1.5 DECC-G

Coevolución Cooperativa basada en Evolución Diferencial (DECC-G, por sus siglas en inglés) [8] es un algoritmo especializado en optimización de problemas de alta

dimensionalidad que utiliza la coevolución cooperativa para conseguir una descomposición del problema original en sub-problemas denominados subcomponentes.

DECC-G se caracteriza por generar subcomponentes aleatoriamente en cada iteración aumentando la probabilidad de optimizar las variables que interactúan entre sí.

2.1.5.1 Coevolución Cooperativa

La coevolución cooperativa (CC) [19] es un marco de trabajo enfocado en la división de un problema y la optimización de cada parte de este por separado. Aprovecha la rápida convergencia de los algoritmos evolutivos en pequeña cantidad de dimensiones, comúnmente entre 100 y 1000 dimensiones. Para cumplir su objetivo utiliza dos estrategias, que son: Optimización de subcomponentes, que optimiza todos los subcomponentes de manera individual y la co-adaptación que escoge genes de varios subcomponentes de manera aleatoria y los evoluciona, aumentando la probabilidad de optimizar genes que estén relacionados.

2.1.5.2 SaNSDE

Evolución diferencial con búsqueda auto-adaptativa de vecindario [20], SaNSDE, es una mejora aplicada al algoritmo clásico de evolución diferencial aplicando una estrategia de búsqueda de vecindario y adaptación de los valores de tasa de cruce y factor de escala cada vez que se obtiene una nueva generación, además utiliza varias técnicas para la mutación, seleccionándolas conforme a los resultados obtenidos en la población anterior, realizando una exploración más variada del espacio de búsqueda.

Los nuevos valores de la tasa de cruce se obtienen dependiendo de las características de la generación anterior y los nuevos valores para el factor de escala se obtienen mediante un valor aleatorio que determina si utiliza una distribución Gaussiana o de Cauchy [20].

2.1.5.3 DE

Evolución diferencial (Differential Evolution, DE) [21], es un algoritmo evolutivo que parte de una población aleatoria con la que intenta abarcar la mayor cobertura del espacio de búsqueda, su característica principal es su proceso de mutación donde se añade la diferencia de pesos entre dos individuos a un tercero. Los individuos mutados son sometidos a un proceso de cruce y un proceso de selección con el correspondiente individuo de la generación anterior para producir un desplazamiento [8].

2.1.6 MOS

El muestreo múltiple de descendientes, MOS [7] es un framework para el desarrollo de algoritmos evolutivos dinámicos, exitoso por combinar diferentes tipo de técnicas en problemas de optimización continua. El concepto de técnica se refiere a un

mecanismo desacoplado del algoritmo principal para generar nuevos individuos, se utilizan simultáneamente y es MOS el encargado de seleccionar entre las técnicas disponibles cuál es la más apropiada de acuerdo al problema y al espacio de búsqueda [7].

MOS originalmente utiliza para la experimentación como técnicas a: Solis and Wets [22][21] y la búsqueda local 1 de trayectorias múltiple para la optimización global a gran escala (MTS-LS1, Multiple trajectory search for large scale global optimization) [23], estos mismas serán utilizadas en el presente trabajo de investigación, sin embargo MOS puede trabajar con cualquier algoritmo y con una cantidad indeterminada de técnicas.

2.1.6.1 Solis and Wets

El algoritmo Solis and Wets [22] es un algoritmo de ascenso a la colina (o Hill Climbing) que utiliza un paso adaptativo de acuerdo a las fallas y éxitos obtenidos durante la ejecución, su funcionamiento consiste en iniciar con una solución aleatoria y generar un vecindario alrededor de está utilizando un vector de sesgos y un vector de desviación.

Este algoritmo realiza un paso adaptativo dependiendo de los fallos y aciertos de las ejecuciones previas. Los valores de los genes de las soluciones obtenidas en el vecindario son generados mediante una distribución gaussiana con media cero (0) y desviación estándar variable, si realiza más de dos aciertos (mejora del mejor) la desviación estándar para la generación del nuevo vecindario se duplica, por el contrario si falla más de una vez (no mejora al mejor) la desviación estándar para la generación del nuevo vecindario se reduce a la mitad [22].

2.1.6.2 MTS-LS1

MTS-LS1 es un algoritmo poblacional que implementa la búsqueda local 1 de [23], haciendo uso de arreglos ortogonales simulados (Simulated orthogonal array, SOA) en la creación de su población inicial para distribuirla de forma uniforme sobre el espacio de soluciones [23].

La búsqueda local 1 de MTS recorre aleatoriamente las variables de cada individuo y realiza modificaciones sobre estas de acuerdo a un rango de búsqueda establecido. El rango de búsqueda parte inicialmente de la mitad del rango de cada variable.

2.1.7 IHDELS

IHDELS (Iterative Hybrid Differential Evolution with Local Search) es una hibridación iterativa del algoritmo de evolución diferencial que integra búsqueda local y aplica reinicios, diseñado para resolver problemas de alta dimensionalidad. Es un algoritmo que combina de manera iterativa un algoritmo basado en evolución diferencial (DE) y utiliza como factor de explotación métodos de búsqueda local (BL) [9].

IHDELS ejecuta en cada iteración el algoritmo SaDE y un método de BL con el objetivo de que se complementen, la BL se selecciona según la calidad de los individuos que produjo en su última ejecución; al ejecutar SaDE o una BL sus parámetros adaptativos inician con los valores de su última ejecución, cuando no se detecta una mejora significativa se aplica un mecanismo de reinicio según sea el caso.

IHDELS utiliza originalmente como BL al algoritmo MTS-LS1 (descrito en la sección **2.1.6.2**) y L-BFGS-B, pese a esto puede utilizar otros algoritmos de búsqueda local.

2.1.7.1 L-BFGS-B

El algoritmo de memoria limitado BFGS con restricciones, es un algoritmo iterativo cuasi-newton que utiliza memoria limitada para problemas de alta dimensionalidad, tiene restricciones de la forma $l_i \leq x_i \leq u_i$ para cada variable de una solución. L-BFGS-B se basa en el método del gradiente y utiliza memoria limitada denotada como B_k para el cálculo de la matriz Hessiana en el punto x_k, consiguiendo que el costo computacional de una iteración del algoritmo se mantenga de orden n [24]. En cada iteración las fases transcendentales del algoritmo son el cálculo generalizado del punto de Cauchy y la minimización del sub-espacio [25].

Dado que el cálculo del gradiente implica el consumo de altas evaluaciones de la función objetivo, este algoritmo no fue utilizado en el presente trabajo y se sustituyó por el algoritmo de ascenso a la colina.

2.1.7.2 Ascenso a la colina

El algoritmo de ascenso a la colina, también conocido como Hill Climbing, es un algoritmo de búsqueda local relacionado con el ascenso del gradiente pero que no implica conocer las derivadas o la matriz Hessiana. Evalúa nuevos individuos en el vecindario del individuo actual y se mueve a uno nuevo si este es mejor que el actual, permitiendo alcanzar un óptimo local [26]. La búsqueda en el vecindario se realiza haciendo alteraciones (mutaciones) en las variables del individuo para obtener una nueva solución.

2.1.8 JMetal

JMetal [27] es un framework especializado en algoritmos meta-heurísticos, desarrollado en java, por el departamento de ciencias de la computación de la Universidad de Málaga. Su fortaleza radica en su flexibilidad y extensibilidad permitiendo que el problema y el algoritmo se encuentran desacoplados, de modo que un algoritmo es capaz de resolver cualquier problema (siempre y cuando cumpla con ciertas restricciones propias del framework). La versión 5.2 de JMetal se compone de 4 módulos:

- jmetal-core: Define la estructura principal para el desarrollo de meta-heurísticas. La ***Figura 3*** resume a brevedad cómo se define un algoritmo, problema y solución en el framework, Jmetal define como *objetivos* los resultados de la

evaluación de una solución, en caso de ser un problema mono-objetivo la lista de objetivos únicamente tendrá un valor. Las *variables* son los diferentes valores que representan a un individuo solución. Los *atributos* son un conjunto de valores adicionales propios del individuo como solución de un problema en particular y las *restricciones* dependiendo del tipo de problema definen cuales son los valores válidos de cada variable de una solución.

- jmetal-algorithm: Contiene todas las meta-heurísticas definidas dentro de JMetal, tanto algoritmos multi-objetivos como mono-objetivos.
- jmetal-problem: Comprende los problemas definidos dentro de JMetal.
- jmetal-exec: Define la ejecución de experimentos o ejecuciones simples de los diferentes algoritmos y problemas en el framework.

La integración de nuevos algoritmos y problemas se facilita gracias a la estructura con la cual fue concebido. Un algoritmo se define usualmente para resolver un tipo de problema, por ejemplo, problemas de valores enteros, binarios o decimales, siendo esta la principal restricción sobre la cual debe definirse un problema para que un algoritmo lo pueda abordar.

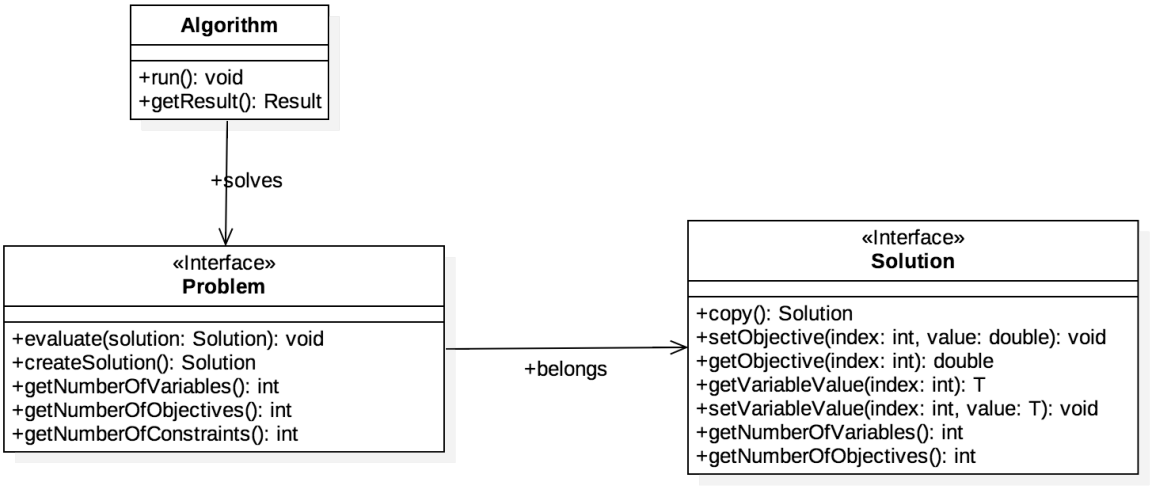

Figura 3 *Arquitectura de JMetal para la definición de un algoritmo y un problema.*

2.2 ESTADO DEL ARTE

La revisión de la literatura señala que para mejorar el rendimiento de una ELM se han propuesto diferentes métodos, los cuales se encuentran en una de las siguientes categorías [4]:

- **Enfoques no evolutivos**: presentan un buen desempeño sin embargo los parámetros de entrada continúan siendo asignados aleatoriamente, como consecuencia la exactitud en la clasificación es baja.

- **Enfoques meta-heurísticos**: son métodos de optimización inspirados en la naturaleza (la mayoría de ellos) y se caracterizan por trabajar con una población de individuos a quienes se les aplican diferentes operaciones, por ejemplo, en los algoritmos genéticos, las operaciones de cruce, mutación y selección.

A continuación, se presentan los trabajos previos más relevantes para el presente proyecto y que se encuentran en el enfoque meta-heurístico.

ELM Evolutivo (E-ELM): E-ELM es un enfoque híbrido propuesto en [28] en el 2005, el cual utiliza evolución diferencial (meta-heurística conocido por su eficiencia en la localización de óptimos globales) para la exploración de los pesos óptimos de entrada y los sesgos de la capa oculta, finalmente para el cálculo analítico de los pesos de salida utiliza la pseudoinversa de Moore-Penrose. E-ELM crea una población con pesos y sesgos asignados aleatoriamente, después realiza el cálculo de los pesos de salida y determina la aptitud (fitness) de cada individuo por medio de una función de costo; posteriormente se realiza la mutación, cruce y selección de los individuos, obteniendo así una nueva población que se evalúa y según su aptitud puede remplazar individuos de la población actual. El proceso se lleva a cabo hasta que se cumple una de las condiciones de parada y el mejor individuo es retornado. E-ELM no requiere que la función de activación sea diferenciable y generalmente logra un mayor rendimiento comparado con algoritmos como BP, GALS [29] y ELM tradicional.

Optimización por enjambre de partículas ELM (PSO-ELM): PSO-ELM descrito en [30] y presentado en el 2006; es un algoritmo híbrido que utiliza optimización por enjambre de partículas (Particle swarm optimization, PSO) para optimizar los pesos de entrada y los sesgos de la capa oculta y finalmente calcula los pesos de salida como lo hace tradicionalmente ELM. PSO-ELM permite que el aprendizaje de una red neuronal se dé en menor tiempo y al igual que el trabajo previo evita la restricción sobre la función de activación en la capa oculta.

Ensamble ELM (EN-ELM): EN-ELM encontrado en [31] y presentado en 2010, utiliza validación cruzada sobre los datos de entrenamiento para evitar el sobreajuste de los parámetros, esto extiende el número de predictores en el ensamblado y garantiza estabilidad y una mejor toma de decisiones. El ensamble se realiza tomando los pesos que están disponibles para cada clasificador del ELM después del entrenamiento. Estos pesos se normalizan mediante la norma de Frobenius, los cuales se ordenan de manera ascendente, la primera mitad de estos pesos se utilizan para tomar decisiones.

Algoritmo genético codificado en números enteros y optimización de enjambre de partículas junto con la máquina de aprendizaje extremo (ICGA-PSO-ELM): Presentado en 2011 [32], utiliza una combinación de un algoritmo genético y el algoritmo de enjambre de partículas acoplados a una ELM para la clasificación de cáncer. El algoritmo utiliza un algoritmo genético codificado en números enteros (ICGA) para seleccionar genes que serán los datos de entrada,

estos genes se toman de todo el conjunto de datos de entrada, los mejores genes son usados para entrenar la red neuronal mediante el algoritmo PSO-ELM que optimiza los pesos de entrada y el número de neuronas en la capa oculta con el gen de datos de entrada obtenido por el ICGA.

Optimización basada en grupos para ELM (GSO-ELM): Es un ELM evolutivo basado en una meta-heurística denominada optimización de búsqueda de grupos (Group Search Optimization) propuesta en 2011 [33]. GSO se basa en el comportamiento social de algunos grupos de animales para optimizar los pesos de entrada y los sesgos de la capa oculta. En este trabajo se utilizaron dos estrategias: la primera creando soluciones y la segunda obteniéndolas de tomar recursos descubiertos por otros. El algoritmo GSO-ELM mostró resultados por encima de todos sus competidores en conjuntos de datos reales.

Búsqueda armónica (Harmony Search, HS) para ELM en predicción de series de tiempo hidrológicas (HS-ELM): El algoritmo HS-ELM fue propuesto en 2012 [34] y define una manera de usar HS junto al algoritmo de entrenamiento ELM para optimizar el número de neuronas de la capa oculta, así como también los pesos entre las conexiones de las neuronas de la capa de entrada y la capa oculta. HS-ELM es utilizado para previsión de revisiones en plantas hidrológicas con resultados prometedores.

Evolucionando ELM con selección adaptativa de operadores y control de parámetros (Evo-ELM): Evo-ELM se presentó en 2013 [35] y es un algoritmo híbrido entre una variante de evolución diferencial y ELM que permite optimizar los parámetros de entrada de una red neuronal. La variante del algoritmo de evolución diferencial selecciona el operador adecuado para la generación de descendientes y ajusta dinámicamente los parámetros de control.

ELM basado en PSO y estrategias de agrupamiento o clustering (CPSOS-ELM): CPSOS-ELM presentado en [36] en el año 2013, es un enfoque evolutivo híbrido que utiliza PSO como búsqueda global con un método de agrupación para optimizar los pesos de entrada y los sesgos de la capa oculta. Utiliza un operador para mejorar el desempeño de PSO y la velocidad de convergencia. CPSOS-ELM demostró obtener mejores resultados de rendimiento en clasificación que el ELM original en 3 de los 4 conjuntos de datos estudiados.

Evolución diferencial para ELM (DE-ELM): DE-ELM es un algoritmo de optimización evolutivo propuesto en 2014 [37] que utiliza validación cruzada para determinar los pesos de entrada óptimos de una ELM. En este trabajo se propone un método de clasificación eficiente para imágenes espectrales basado en ELM. El algoritmo de evolución diferencial (DE) se utiliza para resolver el problema de selección de los pesos de entrada de una ELM. Los resultados del experimento prueban que en la mayoría de los casos DE-ELM provee mayor exactitud de clasificación en relación con los obtenidos por una máquina de soporte vectorial

(Support Vector Machines, SVM), un clasificador reconocido por sus excelentes resultados.

Busqueda armónica diferencial auto adaptativa optimizada para ELM en predicción de series de tiempo financieras (SADHS-OELM): Este trabajo propuesto también en 2014 [38] es un algoritmo de optimización que utiliza la búsqueda armónica (HS) para obtener mejores resultados en las SLFNs, aprovechando la capacidad de generalización de las ELM junto con la capacidad de aprendizaje global del algoritmo HS. Ha sido usado para la selección de los mejores parámetros de la capa oculta como son el sesgo de las neuronas de dicha capa y el factor de regularización de los mínimos cuadrados en el entrenamiento de una SLFN mediante el algoritmo ELM. Fue aplicado en SLFNs de baja complejidad con resultados prometedores.

ELM Memético (M-ELM): Este algoritmo fue propuesto más recientemente en 2015 [4] e incorpora una búsqueda local dentro de una búsqueda global para obtener los parámetros óptimos de una red neuronal utilizando ELM. El algoritmo usa evolución diferencial para realizar la búsqueda global en el espacio de soluciones y aplica optimización local para explotar las zonas más prometedoras. Este trabajo se comparó contra ELM, EN-ELM, V-ELM, SaE-ELM y Evo-ELM y mostró que sus resultados en problemas de clasificación son mejores en términos de exactitud con un tiempo de ejecución competitivo, convirtiéndose en el método del estado del arte para la definición de pesos de la capa de entrada y sesgos de la capa oculta en una SLFN que se entrena con ELM.

Capítulo 3

3. PROPUESTA

3.1 INTRODUCCIÓN

Este capítulo presenta la representación de un individuo solución, las restricciones de un individuo y finalmente la forma como fueron adaptados los tres algoritmos meta-heurísticos para dar solución al problema del entrenamiento de una SLFN utilizando una ELM.

3.2 REPRESENTACIÓN DE UN INDIVIDUO

Con base en lo descrito en la sección **2.1.3** se desea representar los pesos de entrada y sesgos de una SLFN, los cuales serán optimizados, para que a partir de estos la ELM pueda calcular los pesos de salida y la salida de la red neuronal, en otras palabras, se representará a ω y b como parte de un individuo.

Una solución será representada por medio de un vector el cual almacena los valores de los pesos de entrada y sesgos de la capa oculta como se muestra en *la **Figura 4**.* Se toman los pesos entre cada neurona de la capa oculta hacia las neuronas de la capa de entrada y se adiciona el sesgo de la neurona de la capa oculta en cuestión para representar una solución del problema de la ELM. Los valores de un individuo corresponden a valores decimales.

ω_{11}	ω_{12}	$\cdots$	ω_{1m}	b_1	ω_{21}	ω_{22}	$\cdots$	ω_{2m}	b_2	$\cdots$	ω_{n1}	ω_{n2}	$\cdots$	ω_{nm}	b_n

Figura 4 *Representación de un individuo*

Incluir los valores de los sesgos entre los pesos de entrada o dejarlos todos al final del vector no reporta un cambio significativo en la exactitud de los resultados [39]. Pero se toma la representación presentada en ***Figura 4*** ya que en ella quedan en orden todos los pesos y sesgos de la neurona 1, luego de la neurona 2, y así sucesivamente con todas las neuronas de la capa oculta, dando más orden y semántica al individuo.

3.3 RESTRICCIONES DE UN INDIVIDUO

La teoría de la ELM indica que los pesos de entrada y sesgos se conciben al generar valores aleatorios con el rango de $[-1, 1]$, por lo tanto, cada una de las variables que representa un individuo deberá adaptarse a dicho rango y los cambios sobre un individuo deberán contemplar los valores máximos y mínimos establecidos para su correcta modificación.

3.4 EVALUACIÓN DE UN INDIVIDUO

La calidad de una solución se evalúa según la exactitud obtenida durante el proceso de entrenamiento. Si dos soluciones obtienen la misma exactitud, se decide cuál de las dos es mejor con base en la norma de Frobenius de los pesos de salida β, la

cual fue utilizada en [40] para evaluar una solución. Si una solución tiene una norma menor esa es mejor solución. Este procedimiento se realizó solo para los algoritmos DECC-G, MOS e IHDELS ya que el M-ELM no reporta el uso de la norma de Frobenius durante la evaluación de sus individuos.

Por otro lado, dado que existen problemas de clasificación y de regresión la ELM calcula la exactitud según el tipo de problema; en el caso de problemas de clasificación se hace de acuerdo a la cantidad de aciertos obtenidos en el entrenamiento como se expuso anteriormente en la ***Ecuación 6***, lo cual significa que entre mayor sea el valor mejor será una solución. Para los problemas de regresión una ELM calcula el error cuadrático medio entre la solución esperada y la computada por la red neuronal, y en este caso entre menor sea el valor obtenido mejor es el resultado; con esto en mente se decidió unificar los problemas de regresión y clasificación para que en general entre mayor sea el valor obtenido mejor sea la solución (Maximización). Dicho esto, se configuró la ELM implementada para que al calcular la exactitud en problemas de regresión aplique la ***Ecuación 7****,* de la cual se obtienen valores entre $(0,1]$.

$$exactitud = \frac{1}{1 + error}$$

Ecuación 7 *Cálculo de la exactitud en problemas de regresión.*

Así, la exactitud obtenida en problemas de regresión es un valor que entre más se aproxime a 1 indica que la red neuronal quedó mejor entrenada.

Se optó por tomar como parte de la función objetivo la exactitud del entrenamiento, dado que el trabajo M-ELM [4] así lo reporta y se quiere realizar una comparación bajo las mismas condiciones.

3.5 CONJUNTOS DE DATOS

Los conjuntos de datos utilizados en el proyecto de investigación se extrajeron del repositorio de la Universidad de California en Irvine (UCI); la selección de los conjuntos de datos de clasificación se realizó teniendo en cuenta los presentados en [4].

Los conjuntos de datos se procesaron para su uso dentro del proyecto, el cual consistió en eliminar aquellos atributos que no eran necesarios para el aprendizaje de la red neuronal, los valores categóricos se definieron como un número entero y los atributos se normalizaron entre los valores [-1, 1] excepto el atributo de clase.

Finalmente, cada conjunto de datos se dividió en dos archivos, uno para entrenamiento y otro para pruebas, la división se realizó siguiendo la relación presentada en [4] que es aproximadamente 70% de las instancias para entrenamiento y 30% de las instancias para pruebas; no se realizará un análisis de

los conjuntos de datos en ningún subgrupo dado que en el presente trabajo se propone realizar una comparación bajo las mismas condiciones del trabajo [4].

El número de instancias que se utilizaron para entrenamiento y pruebas de cada conjunto de datos puede observarse en la ***Tabla 2*** y en la ***Tabla 3.***

Tabla 2. *División de los conjuntos de datos para problemas de clasificación*

Conjunto de datos	#Entrenamiento	#Pruebas
Banknote	922	450
Blood	500	248
Car	1152	576
Cardiotocography	1426	700
Chart	400	200
ClimateSimulation	324	216
Connectionist	138	70
Contraceptive	1000	473
Dermatology	244	122
Diabetes	576	192
Ecoli	224	112
Fertility	70	30
Glass	154	60
Haberman	204	102
Hayes	88	44
Hill	404	202
Indian	393	190
Ionosphere	234	117
Iris	100	50
Knowledge	269	134
Leaf	230	110
Letter	15000	5000
Libras	260	100
Optdigits	3823	1797
Pen	7494	3498
Planning	109	73
QSARBiodegradation	705	350
Seeds	140	70
Shuttle	43500	14500
Spectf	179	90
Vertebral(2C)	210	100
Vertebral(3C)	210	100
Wdbc	379	190
Wilt	3226	1613
Wine	118	60
Wine(Red)	1066	533
Yeast	989	495
Zoo	70	31

Tabla 3. *División de los conjuntos de datos para problemas de regresión*

Conjunto de datos	#Entrenamiento	#Pruebas
AutoMpg	278	120
AutoPrice	111	48
BodyFat	176	76
Cpu	146	63
Housing	354	152
Sensory	403	173
Servo	116	51
Veteran	95	42

El esquema de cada uno de los archivos de entrenamiento y pruebas se caracteriza por tener un registro en la primera línea del archivo que indique la cantidad de instancias, la cantidad de atributos y el número de clases, separadas entre sí por un espacio; los valores de la clase fueron ubicados al final de cada registro para facilitar la manipulación de los datos.

3.6 INTEGRACIÓN CON EL FRAMEWORK JMETAL

Se definió dentro de JMetal un nuevo módulo denominado *unicauca-ann* el cual enfoca sus esfuerzos en el problema de entrenamiento de SLFN usando ELM y se espera que pueda abordar problemáticas de las redes neuronales en general, de allí su nombre (ANN, Artificial Neural Network).

3.6.1 Definición del problema

El problema de entrenamiento de SLFN con ELM se definió de tipo *Double,* es decir, las variables de una solución son representadas por medio de un arreglo que contiene valores *Double* tal cual se expresó en la sección **3.2**.

El problema de la ELM, representado por la clase *AbstractELMEvaluator* como se muestra en la ***Figura 5*** es la encargada de evaluar una solución, para esto, debe realizar su conversión apropiada a los correspondientes valores de los pesos de entrada y los sesgo de una ELM, una vez estos se identifican se asignan respectivamente a la ELM y esta se entrena, finalmente la exactitud obtenida en el entrenamiento (la cual es calificada con valores entre 0 y 1, siendo 1 el valor más alto, donde todos los valores de entrenamiento fueron acertados) se asigna como el objetivo de índice 0 (mono-objetivo) de la solución estimada como lo indica la ***Ecuación 8***; teniendo en cuenta que el problema fue definido como un problema de maximización.

$$objective[0] = exactitud$$

Ecuación 8 *Evaluación objetivo de una solución.*

El problema se encarga de gestionar los conjuntos de datos sobre los cuales se realizan las evaluaciones, se manejan dos conjuntos de datos separados, uno sobre el cual se realiza el entrenamiento denominado *training_data_set*, que utilizan los

algoritmos para realizar la optimización de los pesos y sesgos de la red neuronal y los datos de evaluación *testing_data_set* que se usan para conocer la exactitud que obtiene la ELM una vez a terminado el proceso de optimización, precisando que como se mencionó previamente, entre los datos de entrenamiento y de evaluación no existe intersección.

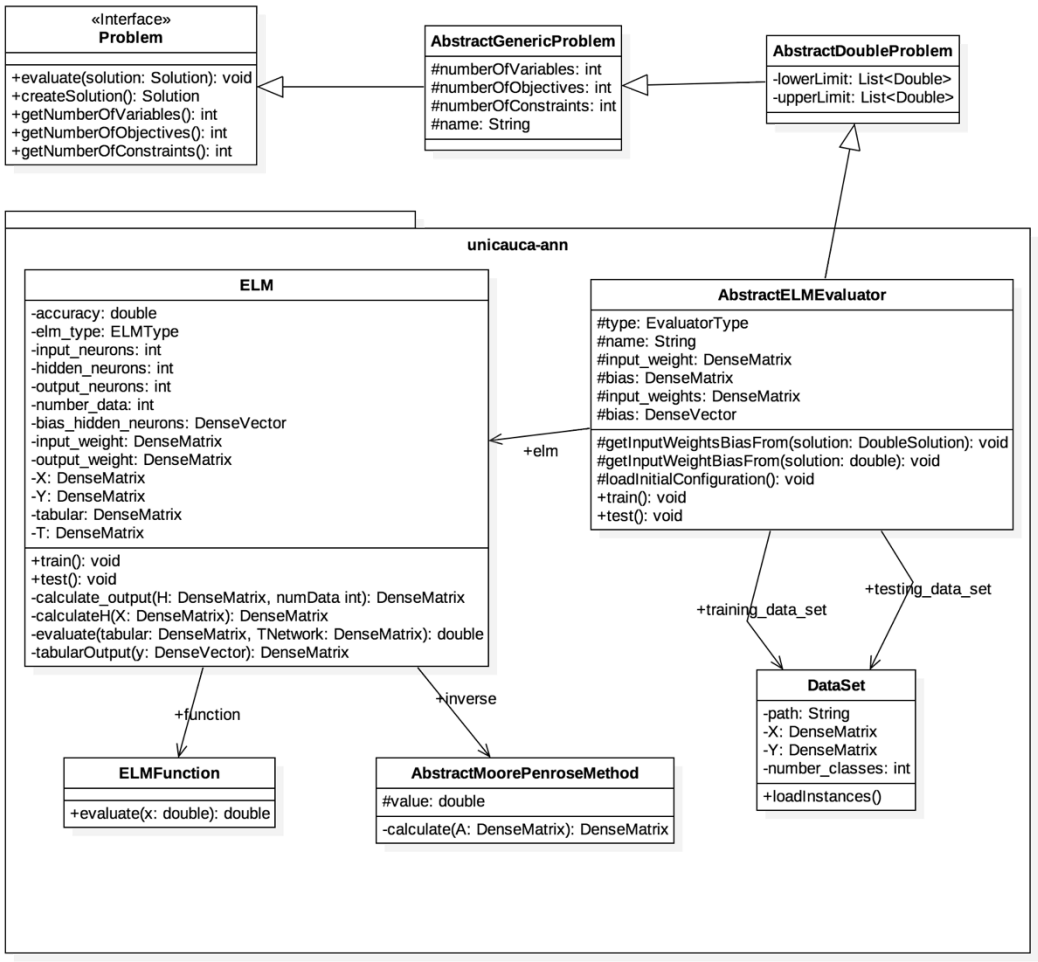

Figura 5 *Diagrama definición del problema ELM en JMetal*

3.6.2 Modelos de validación

Para el presente trabajo de investigación se decidió utilizar dos modelos de validación: entrenamiento-prueba y validación cruzada; las cuales se caracterizan por dividir los conjuntos de datos en dos partes que no se sobreponen entre sí: la primera es utilizada para el entrenamiento del modelo y la segunda para su validación, se denominan conjunto de entrenamiento y conjunto de pruebas respectivamente [41].

La ***Figura 6*** muestra cómo fueron desarrollados los modelos de validación y expone que ambos fueron utilizados en todos los conjuntos de datos tomados para el

proyecto. La figura no muestra todos los conjuntos de datos debido a que quedaría muy extensa, sólo muestra a manera de ejemplo los conjuntos de datos de Iris, Ecoli y Hill. A continuación, se explica en más detalle cada modelo de validación.

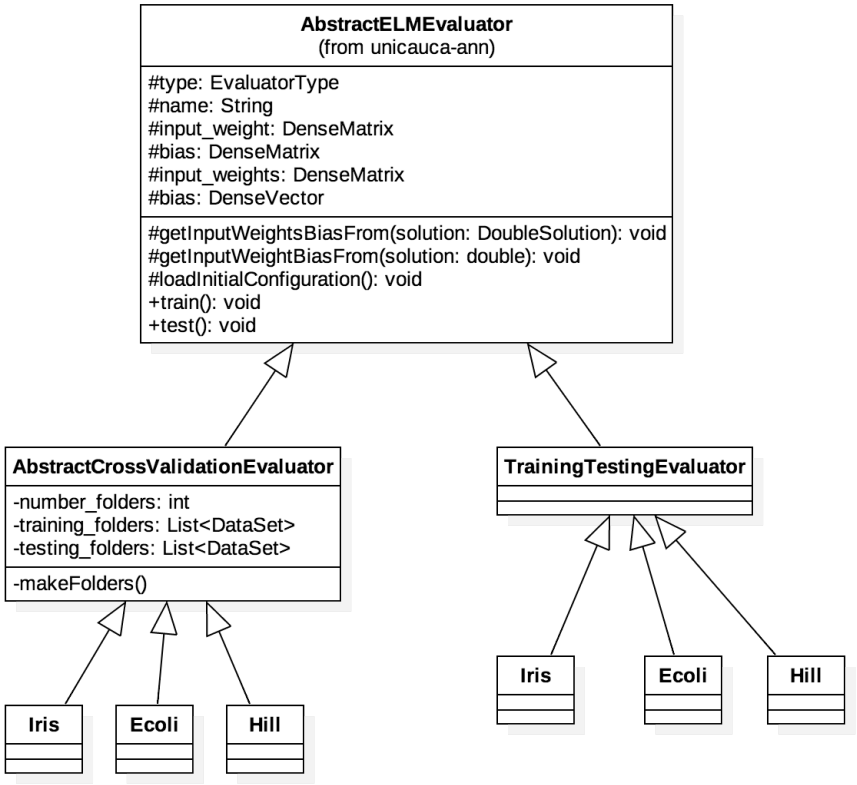

Figura 6 *Diagrama de tipos de problemas*

3.6.2.1 Entrenamiento-prueba

El modelo de entrenamiento-prueba conocido formalmente como método de retención, utiliza únicamente el conjunto de entrenamiento para ajustar el modelo, al finalizar se valida utilizando el conjunto de prueba el cual nunca fue previamente utilizado [42]. La función objetivo se calcula a partir de la exactitud obtenida en el entrenamiento del modelo, es decir: $exactitud = entrenamiento\ ELM$.

3.6.2.2 Validación cruzada con k folders

El modelo de validación cruzada con k folders divide el conjunto de entrenamiento en k subconjuntos similares (o lo más similares posibles) e iterativamente toma cada uno de los k subconjuntos como conjunto de prueba y los otros $k - 1$ subconjuntos se unen para formar el conjunto de entrenamiento [42].

En este caso, el cálculo de la función objetivo se hace a partir del promedio obtenido de las pruebas realizadas sobre los k subconjunto de prueba, como lo indica la **Ecuación 9**. Esto quiere decir que en un problema de validación cruzada una evaluación corresponde realmente a k evaluaciones del modelo ELM.

$$exactitud = \frac{1}{k}\sum_{i=1}^{k} prueba\ ELM$$

Ecuación 9 *Exactitud en validación cruzada.*

3.7 ADAPTACIÓN DE DECC-G

DECC-G es una meta-heurística que se caracteriza por resolver el problema al dividirlo en pequeños sub-problemas. Su funcionamiento está descrito en el ***Seudocódigo 1*** (tomado de [8]) y a continuación, se explica detalladamente su funcionamiento a través de un ejemplo.

Seudocódigo 1. *Meta-heurística DECC_G*

Entrada: Número de subcomponentes $\boldsymbol{subcomponentes} > \mathbf{0}$,
Ciclos $\boldsymbol{ciclos} > \mathbf{0}$,
Evaluaciones para SaNSDE $\boldsymbol{FE} > \mathbf{0}$,
Evaluaciones para DE $\boldsymbol{wFE} > \mathbf{0}$,
Problema a resolver $\boldsymbol{problema}$

Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
población = crearPoblacionAleatoria()
evaluar(población)
n = problema.obtenerNumeroVariables()
s = n / subcomponentes
para i = 1 hasta ciclos
  indices = crearSecuencia(n)
  l = 0
  u = -1
  para j = 0 hasta subcomponentes – 1
    l = u + 1;
    u = l + s – 1
    subLista = indices(l:u)
    poblacion = SaNSDE(población, FE, subLista)
  fin para
  indices = crearSecuencia(subcomponentes, n)
  mejor = encontrarMejor(poblacion)
  aleatorio = encontrarAleatorio(poblacion)
  peor = encontrarPeor(poblacion)
  mejor = DE(wFE, mejor, indices)
  aleatorio = DE(wFE, aleatorio, indices)
  peor = DE(wFE, peor, indices)
fin para
mejor = encontrarMejor(poblacion)
retornar mejor
```

Supóngase que se tiene un problema donde los individuos tienen 12 variables, es decir $n = 12$ (línea 3 en adelante); se definió que habrá tres subcomponentes: $subcomponentes = 3$, entiéndase como subcomponente un subconjunto de las variables de los individuos. La dimensión de cada subcomponente en este caso será de $s = 4$ (línea 4 en adelante).

A continuación, DECC-G crea una permutación de 0 hasta $n - 1$ (línea 6), cada valor corresponde a un subíndice en las variables los individuos, las primeras s variables conforman el primer subcomponente, las segundas s variables el segundo subcomponente y así para todos los valores del vector índices; para efectos del ejemplo supóngase que la permutación obtenida es la siguiente:

$$indices = \{8, 5, 1, 0, 3, 11, 9, 7, 2, 10, 6, 4\}$$

Seguido se crean los subcomponentes, cada uno con s variables tomadas del vector $indices$ distribuidas como se muestra en la ***Figura 7***, donde m corresponde a la cantidad de individuos de la población.

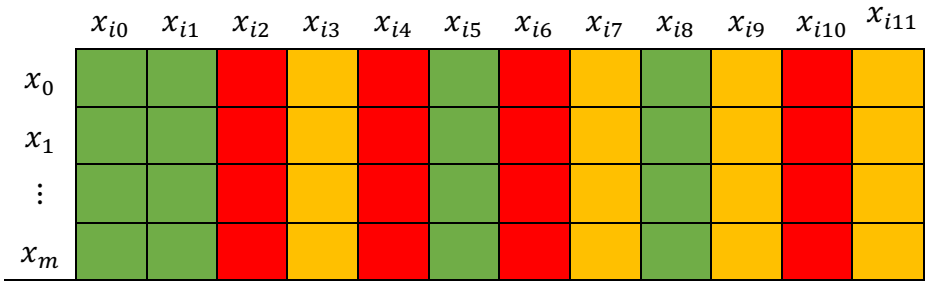

Figura 7 *Representación de los subcomponentes en DECC-G.*

Cada color corresponde a un subcomponente; con esto, aunque SaNSDE evoluciona sobre toda la población que le entrega DECC-G verá a cada individuo como si fuera un individuo con s variables, es decir al evolucionar sobre el primer subcomponente únicamente lo hará sobre las variables resaltadas en color verde y así para el resto de subcomponentes (línea 9 a línea 14).

Al finalizar la evolución sobre los subcomponentes se crea nuevamente una secuencia de índices de tamaño $subcomponente$ con valores entre $[0, n - 1]$ (línea 15). Supóngase que el nuevo vector de índices es el siguiente:

$$indices = \{3,6,1\}$$

DECC-G selecciona tres individuos de su población: el mejor, el peor y uno aleatorio y DE para optimizar cada uno de estos individuos crea una población de pesos con valores entre [0, 1] de m individuos y $subcomponentes$ variables, de esta manera DE evoluciona la población de pesos y la evaluación de cada individuo se realiza como sigue:

Supongase que la ***Figura 8*** es la población de DE y la ***Figura 9*** es el individuo que está optimizando.

	x_{i0}	x_{i1}	x_{i2}
x_0	0.2	0.3	0.7
x_1	…	…	…
$\vdots$	…	…	…
x_m	…	…	…

Figura 8 *Representación del a población de DE en DECC-G.*

x_{i0}	x_{i1}	x_{i2}	x_{i3}	x_{i4}	x_{i5}	x_{i6}	x_{i7}	x_{i8}	x_{i9}	x_{i10}	x_{i11}
0.5	-0.2	0.3	0.7	0.9	0	0.2	-0.3	0.4	-0.1	0.3	-0.8

Figura 9 *Individuo a optimizar en DE.*

Para la evaluación del primer individuo de la población de DE estos pesos deben ser multiplicados con el individuo que se está optimizando, así: cada variable del individuo de pesos deberá multiplicarse con el valor en el individuo a optimizar según el índice que le corresponda en el vector $indices$, en el caso del ejemplo la variable con índice 0 del individuo de pesos se multiplica con la variable de índice 3 del individuo a optimizar.

Los colores de la ***Figura 8*** y ***Figura 9*** corresponden a las variables que deben multiplicarse entre si y da como resultado el individuo de la ***Figura 10***, siendo este el que se evalúa

x_{i0}	x_{i1}	x_{i2}	x_{i3}	x_{i4}	x_{i5}	x_{i6}	x_{i7}	x_{i8}	x_{i9}	x_{i10}	x_{i11}
0.5	-0.14	0.3	0.14	0.9	0	0.06	-0.3	0.4	-0.1	0.3	-0.8

Figura 10 *Multiplicación del individuo con pesos.*

Este procedimiento se realiza para con todos los individuos de la población de DE cuando se está evolucionando un individuo.

3.7.1 Adaptación de evolución diferencial

El algoritmo evolución diferencial se presenta en el ***Seudocódigo 2*** (tomado de [43]) donde la variable $tamañoPoblacion$ (línea 2) corresponde al tamaño de la población y la variable n (línea 2) corresponde a la cantidad de genes de un individuo de la población.

Como se encuentra especificado en [43], la reproducción (línea 4) se realiza para cada individuo de la población, en ella se seleccionan 3 individuos aleatoriamente (x_{i_1}, x_{i_2} y x_{i_3}) de la población actual asegurando que sean diferentes entre sí y diferentes al presente individuo x_i, y aplicando la ***Ecuación 10*** se obtiene el individuo v_i que se conoce como individuo mutado. Además, a estos individuos x_i y v_i se les aplica la ***Ecuación 11*** para obtener el individuo u_i que se conoce como el individuo de prueba o el individuo que competirá contra x_i por pasar a la siguiente

generación. En esta ecuación, $U_j(0,1)$ corresponde a un número aleatorio uniformemente distribuido entre 0 y 1.

Seudocódigo 2. *Algoritmo evolución diferencial.*

Entrada: Máximas evaluaciones $\boldsymbol{maxEvaluaciones} > \mathbf{0}$,
Tasa de cruce $\boldsymbol{CR} > \mathbf{0}$,
Factor de escala $\boldsymbol{F} > \mathbf{0}$
Tamaño población $\boldsymbol{tamanoPoblacion} > \mathbf{0}$
Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
1:    población = crearPoblacionAleatoria(tananoPoblacion)
2:    evaluaciones = tamanoPoblacion
3:    mientras evaluaciones < maxEvaluaciones hacer
4:      poblacionDescendencia = reproduccion(poblacion)
5:      poblacion = remplazo(poblacionDesdendencia)
6:      evaluaciones = evaluaciones + tamanoPoblacion
7:    fin mientras
8:    mejor = encontrarMejor(poblacion)
9:    retornar mejor
```

$$v_i = x_{i_1} + F * \left(x_{i_2} - x_{i_3}\right)$$

Ecuación 10 *Estrategia de reproducción con el esquema DE/rand/1.*

$$u_i(j) = \begin{cases} v_i(j), & \boldsymbol{si}\ U_j(0,1) \leq CR \\ x_i(j), & \boldsymbol{en\ otro\ caso} \end{cases}$$

Ecuación 11 *Aplicación de la tasa de cruce para un individuo en cada gen.*

Finalmente, el reemplazo (línea 5) se realiza al terminar la generación de todos los individuos en la mutación, para este procedimiento se comparan las aptitudes del individuo u_i y x_i. Teniendo en cuenta que los u_i fueron los obtenidos en la reproducción y los x_i las soluciones de la población actual se obtienen los x_i' que serán los que pasan a la siguiente generación. Los x_i' se calculan como se muestra en la ***Ecuación 12*** donde $f(u_i)$ es el valor de aptitud de u_i y $f(x_i)$ es el valor de aptitud de x_i.

$$x_i' = \begin{cases} u_i, & \boldsymbol{si}\ f(u_i) > f(x_i) \\ x_i, & \boldsymbol{en\ otro\ caso} \end{cases}$$

Ecuación 12 *Reemplazo de un individuo.*

3.7.2 Adaptación de SaNSDE

El algoritmo SaNSDE inspirado en evolución diferencial, tiene como objetivo adaptar los valores de factor de escala, tasa de cruce, probabilidad de selección de una estrategia de mutación y la probabilidad de selección del método de generación del factor de escala a medida que pasan las generaciones [20]. El funcionamiento general de SaNSDE corresponde al presentado en DE en el ***Seudocódigo 2***, pero este recibe como parámetros de entrada otra tasa de cruce $\boldsymbol{CR_2} > \mathbf{0}$ y factor de escala $\boldsymbol{F_2} > \mathbf{0}$.

Por otro lado, SaNSDE incluye dos variables adicionales que son la probabilidad $\boldsymbol{p}$ de seleccionar una estrategia de mutación y la probabilidad de control de distribución $\boldsymbol{fp}$. SaNSDE cuenta con dos estrategias de mutación y dependiendo del valor de $\boldsymbol{p}$ se ejecuta una u otra, de esta forma el individuo v_i resultante en la reproducción de SaNSDE se define como se muestra en la ***Ecuación 13***.

$$v_i = \begin{Bmatrix} \textbf{Ecuación 10}, & \boldsymbol{si}\ U_i(0,1) < p \\ \textbf{Ecuación 14}, & \boldsymbol{en\ otro\ caso} \end{Bmatrix}$$

Ecuación 13 *Selección de la estrategia de reproducción.*

$$v_i = x_i + F * (x_{best} - x_i) + F * (x_{i_1} - x_{i_2})$$

Ecuación 14 *Estrategia de reproducción con el esquema DE/current to best/1.*

La ***Ecuación 14*** requiere que se seleccionen aleatoriamente 2 individuos (x_{i_1}, x_{i_2}) de la población diferentes entre sí y diferentes al individuo actual (x_i), un cuarto individuo que es el individuo con mejor (x_{best}) valor de aptitud de la población seleccionada y un factor de escala $\boldsymbol{F_2}$ el cual es un parámetro auto-adaptativo.

La ***Ecuación 10*** utiliza como factor de escala el parámetro $\boldsymbol{F}$ pero en SaNSDE se renombra por $\boldsymbol{F_1}$ y es auto-adaptativo.

Al finalizar el periodo de aprendizaje, la probabilidad p se actualiza como se muestra en la ***Ecuación 15*** donde $\boldsymbol{ns1}$ es el número de individuos generados por la ***Ecuación 10*** y agregados a la siguiente generación, $\boldsymbol{ns2}$ es el número de individuos generados por la ***Ecuación 14*** y agregados a la siguiente generación, $\boldsymbol{nf1}$ es el número de individuos generados por la ***Ecuación 10*** y descartados en la siguiente generación y $\boldsymbol{nf2}$ es el número de individuos generados por la ***Ecuación 14*** y descartados en la siguiente generación, todos estos valores inician en 0 y al finalizar el cálculo de $\boldsymbol{p}$, los valores de $\boldsymbol{ns1}, \boldsymbol{ns2}, \boldsymbol{nf1}$ y $\boldsymbol{nf2}$ son nuevamente fijados a 0.

$$p = \frac{ns1 * (ns2 + nf2)}{ns2 * (ns1 + nf1) + ns1 * (ns2 + nf2)}$$

Ecuación 15 *Actualización de la probabilidad p.*

Los factores de escala se adaptan para cada descendiente generado en la reproducción y este se obtiene como muestra la ***Ecuación 16*** donde $N_i(0.5, 0.3)$ es

una distribución normal con media 0.5 y desviación estándar 0.3, $U_i(0,1)$ es un valor aleatorio entre 0 y 1 con una distribución uniforme, $\boldsymbol{fp}$ es la probabilidad de escoger una estrategia de generación del nuevo valor de $\boldsymbol{F}$, y $\boldsymbol{\delta_i}$ es un valor obtenido de la distribución de Cauchy.

$$F_i = \begin{cases} N_i(0.5, 0.3), & \boldsymbol{si}\ U_i(0,1) < fp \\ \delta_i, & \boldsymbol{en\ otro\ caso} \end{cases}$$

Ecuación 16 *Selección del método de cálculo del nuevo factor de escala.*

Al finalizar el periodo de aprendizaje el nuevo valor de $\boldsymbol{fp}$ se calcula como se muestra en la ***Ecuación 17*** donde $\boldsymbol{fpns1}$ es el número de individuos cuyo factor de escala fue generado con la distribución normal y fueron agregados a la siguiente generación, $\boldsymbol{fpns2}$ es el número de individuos cuyo factor de escala fue generado con la distribución de Cauchy y fueron agregados a la siguiente generación, $\boldsymbol{fpnf1}$ es el número de individuos cuyo factor de escala fue generado con la distribución normal, pero no fueron agregados a la siguiente generación, $\boldsymbol{fpnf2}$ es el número de individuos cuyo factor de escala fue generado con la distribución de Cauchy, pero no fueron agregados a la siguiente generación, todos estos valores inician en 0 y al finalizar el cálculo de $\boldsymbol{fp}$ los valores de las variables $\boldsymbol{fpns1}, \boldsymbol{fpns2}, \boldsymbol{fpnf1}$ y $\boldsymbol{fpnf2}$ son devueltos a 0.

$$fp = \frac{fpns1 * (fpns2 + fpnf2)}{fpns2 * (fpns1 + fpnf1) + fpns1 * (fpns2 + fpnf2)}$$

Ecuación 17 *Actualización de la probabilidad fp.*

El cruce en SaNSDE se realiza como se especifica en la ***Ecuación 11***, la tasa de cruce al igual que el factor de escala, se obtienen para cada individuo en el proceso de reproducción, cada uno de los valores generados para la tasa de cruce son almacenados en un vector denominado CR_{rec}, de la misma manera, la diferencia de aptitud entre el individuo actual y el individuo generado se almacena en un vector denominado $\boldsymbol{f_{rec}}$ donde $f_{rec}(k) = f(k) - f_{new}(k)$ y el nuevo valor de $\boldsymbol{CR}$ es calculado como se muestra en la ***Ecuación 18***.

$$CR_i = N_i(CRm, 0.1)$$

Ecuación 18 *Valor de la tasa de cruce en la siguiente generación.*

$$CRm = \sum_{k=1}^{|CR_{rec}|} w_k * CR_{rec}(k)$$

Ecuación 19 *Valor de CRm en la siguiente generación.*

$$w_k = f_{rec}(k) / \left(\sum_{k=1}^{|f_{rec}|} f_{rec}(k) \right)$$

Ecuación 20 *Valor de wk para cada valor de CRrec.*

Al terminar el cálculo de $\boldsymbol{CR}$ para todos los individuos de la generación actual, los valores de los vectores $\boldsymbol{CR_{rec}}$ y $\boldsymbol{f_{rec}}$ se eliminan. Finalmente se realiza el reemplazo de los individuos haciendo uso de la ***Ecuación 12*** comparando todos los individuos de la población con todos los individuos generados en la reproducción y se actualizan las evaluaciones.

3.8 ADAPTACIÓN DE MUESTREO DE MÚLTIPLES DESCENDIENTES

La meta-heurística MOS fue adaptada a JMetal y su funcionamiento general se encuentra descrito en el ***Seudocódigo 3*** (tomado de [7]).

Seudocódigo 3. *Meta-heurística MOS.*

Entrada: Máximas evaluaciones $\boldsymbol{maxEvaluaciones} > \mathbf{0}$,
Evaluaciones por iteración $\boldsymbol{FE} > \mathbf{0}$,
Técnicas $\boldsymbol{tecnicas} = \{\boldsymbol{t_0}, \boldsymbol{t_1}, \dots, \boldsymbol{t_n}\}$
Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

1: `evaluaciones = 0`
2: `n = |tecnicas|`
3: $\forall\, j\; radio_participacion_j = \frac{1}{n}$
4: `mejor = ejecutarTecnicas(radio_participacion, FE, mejor)`
5: `evaluaciones = evaluaciones + FE`
6: **`mientras`** `evaluaciones < maxEvaluaciones` **`hacer`**
7: $\forall\, j\; calidad_j = \frac{1}{|individuos_j|} \sum_{i=1}^{|individuos_j|} individuos_j[i].objetivo$
8: $\forall\, j\; radio_participacion_j = PF(calidad_j)$
9: `mejor = ejecutarTecnicas(radio_participacion, FE, mejor)`
10: `evaluaciones = evaluaciones + FE`
11: **`fin mientras`**
12: **`retornar`** `mejor`

El algoritmo recibe como parámetros el número máximo de evaluaciones que se desean ejecutar para realizar la optimización del problema, la máxima cantidad de evaluaciones cada vez que las técnicas se ejecutan y las técnicas que se van a emplear. Entiéndase a las técnicas como los algoritmos subordinados a utilizar. MOS se encarga de determinar cuál de las técnicas produce mejores individuos a través del tiempo, otorgándole a la mejor técnica un porcentaje de participación más alto en cada iteración.

El radio de participación se refiere al porcentaje de evaluaciones de la función objetivo que cada técnica puede realizar en la iteración actual; en la línea 3 del ***Seudocódigo 3*** se dividen los radios de participación en las n técnicas a utilizar, dado que es la primera iteración, todas tienen el mismo porcentaje de participación.

Las técnicas se ejecutan una después de la otra utilizando como entrada el mejor individuo generado por la técnica anterior; la primera técnica que es ejecutada no recibe ningún individuo como entrada y se encarga de generar según sus características el primer individuo. Cuando MOS utiliza simultáneamente técnicas poblacionales y no poblacionales, las técnicas poblaciones deberán asumir al mejor individuo generado por la técnica anterior como el primer individuo de su población y generar el resto de individuos según le corresponda, mientras que la técnica no poblacional recibe de la poblacional el mejor individuo que este encontró y evoluciona a partir de él.

La calidad de una técnica se computa como el promedio de las evaluaciones de todos los individuos generados por la técnica, como se describe en la línea 7 del ***Seudocódigo 3***.

Los radios de participación de una técnica son actualizados según la calidad de los individuos obtenidos en la iteración anterior [7], aplicando la función de participación de la ***Ecuación 21***.

$$PF(calidad_j) = \begin{cases} radio_participacion_j + \eta & \boldsymbol{si} \quad j \in \Omega \\ radio_participacion_j - \Delta_j & \boldsymbol{en\ otro\ caso} \end{cases}$$

Ecuación 21 *Función de participación técnicas de MOS [7].*

Donde Ω corresponde al índice de las técnicas que obtuvieron la mejor calidad en la iteración anterior:

$$\Omega = \{\, l \mid calidad_{\,l} \geq calidad_{\,m} \ \forall\, l, m \in [1, n] \,\}$$

Ecuación 22 *Técnicas con mejor calidad MOS [7].*

$$calidad_{mejor} = \max\{\, calidad_j \ \ \forall\, j \in [1, n]\}$$

Ecuación 23 *Mejor calidad técnicas de MOS [7].*

Además, Δ_j representa el decremento de participación de una técnica, excepto para las mejores técnicas, el cual se computa como la diferencia relativa entre la mejor calidad y la calidad de una técnica j [7], como lo indica la ***Ecuación 24***, donde ξ es un factor de reducción.

$$\Delta_j = \xi * \frac{calidad_{mejor} - calidad_j}{calidad_{mejor}} * radio_{participacion_{\,j}} \ \forall\, j \in [1, n] \mid j \notin \Omega$$

Ecuación 24 *Cálculo del decremento de participación de una técnica MOS [7].*

Y η es el incremento de participación de las mejores técnicas y se obtiene de la ***Ecuación 25***.

$$\eta = \frac{\sum_{k \notin \Omega} \Delta_k}{|\Omega|}$$

Ecuación 25 *Incremento de participación de las mejores técnicas de MOS [7].*

El presente trabajo utilizó cómo técnicas a MTS-LS1 (poblacional) y Solis and Wets (no poblacional), no obstante, MOS puede ser utilizado con cuales quiera otros algoritmos y no se restringe al manejo de sólo dos técnicas, la integración de MOS a JMetal se realizó con esto en mente, de modo que cualquier algoritmo puede ser utilizados como técnicas dentro de MOS, para conseguirlo MOS no recibe directamente los algoritmos, sino que un algoritmo se envuelve dentro de una Técnica que dadas las características del algoritmo (como si es o no poblacional) se encarga de realizar su configuración a partir del individuo inicial que se le entrega para que realice su evolución.

La ***Figura 11*** presenta el diagrama UML de la implementación que se realizó para que MOS pueda utilizar cualquier algoritmo de optimización como técnica, la clase *Technique* se define como abstracta de modo que la manera de evolucionar de las técnicas depende del algoritmo que esté implementando, cada instancia de una técnica es especializada en realizar la ejecución de su algoritmo. Así para integrar nuevos algoritmos dentro de MOS únicamente se debe crear la técnica especializada en evolucionar dicho algoritmo.

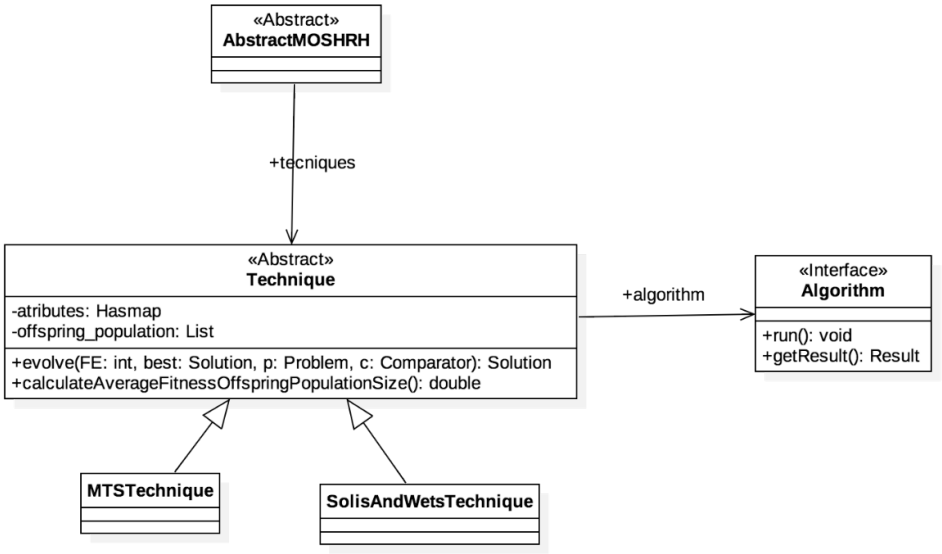

Figura 11 *Implementación técnica de MOS.*

La evolución de una técnica recibe: la cantidad de evaluaciones de la función objetivo que se le asignó según su radio de participación, la mejor solución

encontrada hasta el momento para evolucionar a partir de ella, el problema a resolver y un comparador para determinar si se está maximizando o minimizando en el problema. La evolución de una técnica retornará siempre la mejor solución encontrada.

Otra característica de las técnicas es que, al terminar su ejecución, deberá guardar cuales fueron los individuos que generó el algoritmo durante su evolución, de los cuales se calcula posteriormente el promedio de las evaluaciones de dichos individuos.

3.8.1 Adaptación de Solis And Wets

El algoritmo Solis and Wets se adaptó a JMetal para utilizarse como una de las estrategias que utiliza MOS en la optimización y su funcionamiento es el presentado en el ***Seudocódigo 4*** (tomado de [21]).

Como primer paso se inicializa el vector $\boldsymbol{bias}$ con ceros en todas sus dimensiones, este vector tendrá la misma cantidad de dimensiones como genes tenga el individuo y se utiliza para guiar las nuevas soluciones hacia donde se haya encontrado la mejor solución; también se inicializan los $\boldsymbol{fallos}$ y $\boldsymbol{aciertos}$ en cero y se crea una solución inicial con valores aleatorios.

Solis and Wets funciona como un algoritmo de ascenso a la colina, pero la longitud del avance es adaptativa y se realiza en dos pasos: el primero se encuentra en la línea 6 del ***Seudocódigo 4*** que es generar un vecindario, la forma de generarlo se muestra en el ***Seudocódigo 5*** (tomado de [21]) y para la generación de los vecinos se utilizan la ***Ecuación 26*** y la ***Ecuación 27***.

$$\mathrm{x}_\mathrm{i}' = \mathrm{x}_\mathrm{i} + \mathrm{b}_\mathrm{i} + \mathrm{d}_\mathrm{i}$$

Ecuación 26 *Generación del nuevo individuo por suma de la longitud del avance.*

$$x_i' = x_i - b_i - d_i$$

Ecuación 27 *Generación del nuevo individuo por resta de la longitud del avance.*

Donde x' es un nuevo vecino, x es el individuo del cual se genera el vecino, b es el vector bias y d es el vector de desviación.

Luego se obtiene el mejor individuo generado en el vecindario y se procede a comparar la aptitud de este contra la aptitud del mejor individuo como se muestra en la línea 6 del ***Seudocódigo 4***, en caso de ser mayor la aptitud del nuevo individuo generado, se actualiza el vector $\boldsymbol{bias}$ dependiendo de la estrategia que haya sido utilizada para generar el individuo, si se utilizó la ***Ecuación 26***, $\boldsymbol{bias}$ se actualiza como se muestra en la ***biasi***$= (0.2 * bias_i) + \left(0.4 * \left(desviacion_{del_{individuo\ i}} + bias_i\right)\right)$

Ecuación 28.

$$bias_i = (0.2 * bias_i) + \left(0.4 * \left(desviacion_{del_{individuo\,i}} + bias_i\right)\right)$$

Ecuación 28 Actualización del vector bias por suma.

Si se utilizó la ***Ecuación 27***, $bias$ se actualiza como se muestra en la ***Ecuación 29***.

$$bias_i = bias_i - (0.4 * (desviacion_del_individuo_i + bias_i))$$

Ecuación 29 *Actualización del vector bias por resta.*

Seudocódigo 4 *Algoritmo Solis And Wets.*

Entrada: Máximas evaluaciones $\boldsymbol{maxEvaluaciones > 0}$,
Desviación estándar $\boldsymbol{rho > 0}$,
Tamaño del vecindario $\boldsymbol{cantidad_vecinos > 0\ y\ par}$

Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
bias = 0⃗
fallos = 0, aciertos = 0
mejor = crear_individuo()
evaluaciones = 0
mientras evaluaciones < maxEvaluaciones hacer
  generar_vecindario(mejor)
  mejor_del_vecindario = encontrar_mejor_en_vecindario()
  si aptitud(mejor_del_vecindario)>aptitud(mejor) entonces
     actualizar_bias()
     mejor = mejor_del_vecindario
     aciertos++
     fallos = 0
  sino
     fallos++
     aciertos = 0
  fin si
  si aciertos > 2 entonces
     rho = rho * 2
     aciertos = 0
  fin si
  si fallos > 1 entonces
     rho = rho / 2
     fallos = 0
     si rho < 1 entonces
        rho = 1
     fin si
  fin si
  evaluaciones = evaluaciones + (cantidad_vecinos * 2)
fin mientras
retornar mejor
```

Y se sustituye el mejor individuo por el mejor individuo encontrado en el vecindario, así como también los fallos vuelven a cero y se registra un acierto, línea 10, 11 y 12 del ***Seudocódigo 4***.

En caso contrario, al no encontrar un individuo con un valor de aptitud mayor que el mejor, se registra un fallo y los aciertos se vuelven a cero, línea 14 y 15 del ***Seudocódigo 4***.

Si se registran más de dos aciertos seguidos, la desviación estándar aumenta al doble y los aciertos vuelven a cero; línea 18 y 19 del ***Seudocódigo 4***, si se registra más de un fallo seguido, la desviación estándar disminuye a la mitad y los fallos vuelven a cero; línea 22 y 23 del ***Seudocódigo 4***.

Seudocódigo 5 *Generación de vecinos en Solis and Wets.*

Entrada: Cantidad de vecinos a generar $\boldsymbol{cantidadVecinos > 0\ y\ par}$, Individuo al que se quiere generar el vecindario $\boldsymbol{individuo}$

Salida: El vecindario generado $\boldsymbol{vecindario}$

```
1:   vecinos_generados = 0
2:   mientras vecinos_generados < cantidad_vecinos hacer
3:      desviación = obtener_desviacion()
4:      vecino1_i = (individuo_i + bias_i + desviacion_i)
5:      vecino2_i = (individuo_i − bias_i − desviacion_i)
6:      vecindario.agregar(individuo1)
7:      vecindario.agregar(individuo2)
8:      vecinos_generados += 2
9:   fin mientras
10:  retornar vecindario
```

El procedimiento $obtener_desviacion()$ devuelve un vector del mismo tamaño como genes tenga el individuo y cada uno de sus valores es generado mediante una distribución normal con media 0.5 y desviación estándar $\boldsymbol{rho}$.

3.8.2 Adaptación de MTS-LS1

El algoritmo MTS-LS1 es poblacional y realiza la evolución de sus individuos a través de una búsqueda local. La población inicial se hace con base en un SOA los cuales se edifican a partir de una secuencia c que contiene una permutación aleatoria de $0, 1, \ldots m-1$ donde m es el total de individuos de la población. La matriz $SOA_{m\,x\,n}$ (donde n corresponde al total de atributos de una solución) se llena seleccionando secuencialmente los valores en c y ubicándolos en la primera columna del $SOA_{m\,x\,n}$ donde se selecciona una celda aleatoria y vacía de la dicha columna, este proceso se aplica secuencialmente a las columnas del $SOA_{m\,x\,n}$; para proveer más claridad la construcción del SOA se explica en el ***Seudocódigo 6*** (tomado de [23]).

Seudocódigo 6 *Construcción SOA.*

Entrada: Total individuos $\boldsymbol{m} > \mathbf{0}$,
Total atributos de las soluciones $\boldsymbol{n} > \mathbf{0}$
Salida: Arreglo ortogonal simulado $\boldsymbol{SOA_{m\,x\,n}}$

```
1:    c = crearSecuencia(m)
2:    para i = 1 hasta m
3:      j = obtenerFilaVaciaEnColumna(i,SOA)
4:      SOA[i][j] = c[i]
5:    fin para
6:    retornar SOA
```

El algoritmo MTS-LS1 detallado en el ***Seudocódigo 7*** (tomado de [23]) construye un SOA, con base en él se asignan los valores de cada individuo, teniendo en cuenta las restricciones que tiene cada variable de la solución según lo indica la línea 5 del ***Seudocódigo 7*** y los individuos son evaluados. De cada individuo se sabrá si ha sido o no mejorado por la búsqueda local, al inicio se asume que todos los individuos han sido mejorados. Dada la naturaleza de los diferentes problemas se puede tener que los límites superiores e inferiores de una solución varíen entre sus variables, así se establece un rango de búsqueda por cada variable de cada individuo como el promedio entre su límite superior y límite inferior, estos rangos de búsqueda irán variando con el tiempo.

La condición de parada se hará cuando se alcancen las máximas evaluaciones permitidas, en cada iteración se recorrerán todos los individuos. La búsqueda local retornará un grado el cual determina numéricamente que tan buenas fueron las mejoras sobre el individuo.

Es importante resaltar que las evaluaciones se contarán dentro de la búsqueda local ya que cada individuo al que se le aplica la búsqueda local realiza un número diferente de evaluaciones según diferentes casos.

La búsqueda local 1 detallada en el ***Seudocódigo 8*** (tomado de [23]), utiliza el rango de búsqueda de las variables del individuo y realiza cambios sobre estas aleatoriamente. Se consideran los siguientes escenarios: 1) el cambio a la variable creó un mejor individuo sobre la población, 2) el individuo mejoró, pero no superó al mejor; en estos escenarios significa que puede continuar a la modificación de la siguiente variable y 3) empeoró el individuo, en cuyo caso se realiza un nuevo cambio sobre la variable para darle la oportunidad de mejora. En caso haber mejorado un individuo se aumentará el grado de mejora, el cual indica qué tan buenas fueron las modificaciones sobre el individuo. La principal característica de esta búsqueda local es que, por cada variable del individuo, se realiza un cambio

sobre una variable y se mide los efectos de dicho cambio; los cambios de una variable se hacen únicamente teniendo en consideración los rangos de búsqueda del individuo.

Seudocódigo 7 *Algoritmos MTS-LS1.*

Entrada: Total individuos $\boldsymbol{m} > \mathbf{0}$,
Total atributos de las soluciones $\boldsymbol{n} > \mathbf{0}$,
Límites inferiores de las variables de una solución $\boldsymbol{l}$,
Límites superiores de las variables de una solución $\boldsymbol{u}$,
Máximas evaluaciones $\boldsymbol{maxEvaluaciones} > \mathbf{0}$

Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
SOA_m x n = construirSOA(m, n)
para i = 1 hasta m
  para j = 1 hasta n
         x[i][j] = l[i] + (u[i] – l[i]) * SOA[i][j] / (m – 1)
  fin para
fin para
mejor = seleccionarMejor(x)
para i = 1 hasta m
  mejorado[i] = verdadero
  para j = 1 hasta n
    rangoBusqueda[i][j] = (u[i] – l[i])/2
  fin para
fin para
evaluaciones = m
mientras evaluaciones < maxEvaluaciones hacer
  para i = 1 hasta m
      BusquedaLocal1(x[i], rangoBusqueda[i])
  fin para
fin mientras
mejor = seleccionarMejor(x)
retornar mejor
```

Seudocódigo 8 *Búsqueda local 1.*

Entrada: Individuo $\boldsymbol{x_k}$
Rango de búsqueda $\boldsymbol{rangoBusqueda}$
Salida: Grado de las mejoras hechas al individuo $\boldsymbol{grado}$

```
si mejorado[k] = falso entonces
  para j = 1 hasta n
    rangoBusqueda[k][j] = rangoBusqueda[i][j] /2
    si rangoBusqueda[k][j] < 1e-3 entonces
      rangoBusqueda[k][j] = (u[k] – l[k]) * 0.4
    fin si
  fin para
fin si
mejorado[k] = falso
grado = 0
secuencia = crearSecuencia(n)
para j = 1 hasta n
  i = secuencia(j)
  x_k[i] = x_k[i]  - rangoBusqueda[k][i]
  si x_k es mejor que el mejor individuo entonces
    mejor = x_k
    grado = grado + BONUS_1
  fin si
  si la evaluación x_k es la misma entonces
    restaurar el valor original de x_k
  si no
    si la evaluación x_k empeora entonces
      restaurar el valor original de x_k
      x_k[i] = x_k[i]  + (rangoBusqueda[k][i] * 0.5)
      si x_k es mejor que el mejor individuo entonces
         mejor = x_k
         grado = grado + BONUS_1
      fin si
      si la evaluación x_k no ha mejorado entonces
         restaurar el valor original de x_k
      si no
         grado = grado + BONUS_2
         mejorado[k] = verdadero
      fin si
    si no
      grado = grado + BONUS_2
      mejorado[k] = verdadero
    fin si
```

```
39:     fin si
40:   fin para
41:   retornar grado
```

3.9 ADAPTACION DE IHDELS

IHDELS es una meta-heurística que iterativamente aplica un algoritmo basado en DE como componente exploratorio y un algoritmo de búsqueda local para la explotación; en el ***Seudocódigo 9*** (tomado de [9]) se describe más detalladamente los componentes del algoritmo.

IHDELS parte de una población aleatoria y de un individuo inicial que se construye con las restricciones superiores e inferiores de cada variable del problema. La selección de la búsqueda local se hace como sigue: inicialmente todas las búsquedas locales son ejecutadas como se indica en la línea 5 del ***Seudocódigo 9*** y se almacena para cada una la tasa de mejora, la cual se define con la ***Ecuación 30*** [9]. En cada iteración se aplica aquella búsqueda local con mayor tasa de mejora.

$$TasaMejora_{BL} = \frac{evaluacionAnterior - evaluacionNueva}{evaluacionAnterior}$$

Ecuación 30 *Tasa de mejora de una búsqueda loca enl IHDELS.*

IHDELS mantiene un registro de la mejor solución encontrada a la cual se le denomina $\boldsymbol{mejor}$, por otro lado, existe una solución llamada $\boldsymbol{mejorActual}$ la cual es utilizada para el cálculo de la tasa de mejora y para guiar la búsqueda en la iteración actual. IHDELS define un umbral que permite identificar cuando la tasa de mejora ha sido significativa, para que esto suceda la tasa de mejora debe ser mayor o igual al valor del umbral [9].

IHDELS define un mecanismo de reinicio de los parámetros adaptativos de las BL o de la población en los siguientes casos [9]:

- Si al aplicar DE y la BL durante $\boldsymbol{reinicios}$ iteraciones inmediatamente anteriores, la tasa de mejora no ha sido lo suficiente, el algoritmo se reinicia, lo cual significa: 1) que el individuo $\boldsymbol{mejorActual}$ es alterado como lo indica la línea 25 del ***Seudocódigo 9*** donde $\boldsymbol{rand_i}$ devuelve un número aleatorio entre $[-0.05, 0.05]$; 2) la población se reinicia aleatoriamente y 3) los parámetros adaptativos de las BL vuelven a su valor original.
- Si tras aplicar DE no hay una mejora, se reinicia aleatoriamente la población.
- Si en la ejecución de la BL no mejora nada, se reinician sus parámetros adaptativos.

El presente trabajo utiliza como BL a MTS-LS1 y Ascenso a la colina; su implementación se realizó teniendo en cuenta que las BL pueden variar tanto en definición como en cantidad. La ***Figura 12*** es una representación de la integración de una BL a JMetal; cada BL se encarga de calcular su tasa de mejora, asignarla durante cada evolución y optimizar a partir de la población que se le entrega,

adicionalmente dado que cada búsqueda local maneja diferentes parámetros adaptativos cada una debe realizar el reinicio de estos según le corresponda.

Seudocódigo 9 *Meta-Heurística IHDELS.*

Entrada: Máximas evaluaciones $\boldsymbol{maxEvaluaciones} > \mathbf{0}$
Búsquedas locales $\boldsymbol{busquedasLocales.tamanio} > \mathbf{0}$
Número reinicios $\boldsymbol{reinicios} > \mathbf{0}$
Tamaño población $\boldsymbol{tamanioPoblacion} > \mathbf{0}$
Número de evaluaciones de DE $\boldsymbol{FE_DE} > \mathbf{0}$
Número de evaluaciones búsqueda local $\boldsymbol{FE_LS} > \mathbf{0}$
Valor umbral $\mathbf{0} < \boldsymbol{umbral} < \mathbf{1}$
Dominio de búsqueda $\boldsymbol{a} < \boldsymbol{b}$

Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
evaluaciones = 0
población = crearPoblacionAleatoria(tamanioPoblacion)
evaluarPoblacion(población)
solucionInicial = (limiteSuperior + limiteInferior) / 2
mejorActual = aplicarBusquedasLocales(solucionInicial)
mejor = mejorActual
contadorBL = 0
mientras evaluaciones < maxEvaluaciones
   ant = mejorActual.evaluacion
        mejorActual = SaDE(mejorActual,población,FE_DE)
   mejora = (ant – mejorActual.evaluacion) / ant
   si mejora = 0 entonces
      población = crearPoblacionAleatoria(tamanioPoblacion)
      evaluarPoblacion(población)
    fin si
  BL = seleccionarBL()
  mejorActual = BL(población, FE_BL, mejorActual)
  mejor = seleccionarMejor(mejor, mejorActual)
  si BL.tasaMejora = 0 entonces
      BL.reiniciarParametros()
  fin si
  si BL.tasaMejora < umbral entonces
     contadorBL = contadorBL + 1
      si contadorBL  = reinicios entonces
        mejorActual[i] = mejor[i] + rand_i * 0.1 (b – a)
        población = crearPoblacionAleatoria(tamanioPoblacion)
        evaluarPoblacion(población)
        busquedasLocales.reiniciarParametros()
        contadorBL = 0
      fin si
  si no
```

```
32:         contadorBL = 0
33:      fin si
34:   fin mientras
35:   retornar mejor
```

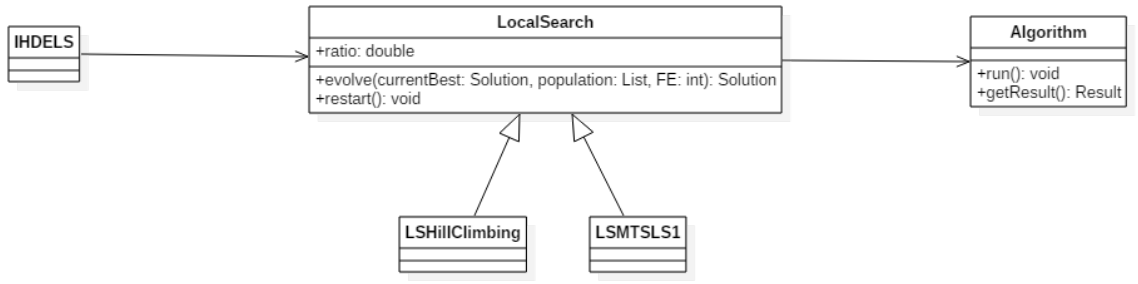

Figura 12 *Implementación de las búsquedas locales en IHDELS.*

Las búsquedas locales utilizadas por este proyecto de investigación fueron MTS-LS1 y Ascenso por la colina; MTS-LS1 como búsqueda local toma la solución $\boldsymbol{mejorActual}$ como el mejor individuo encontrado y empieza la optimización a partir de los individuos de la población que se le indican. Ascenso a la colina al no ser un algoritmo poblacional divide las evaluaciones que se le han asignado entre cada uno de los individuos de la población y los optimiza por separado, finalmente selecciona el mejor individuo de la población y lo compara contra $\boldsymbol{mejorActual}$ retornando el mejor entre los dos.

3.9.1 Adaptación de ascenso a la colina

El algoritmo ascenso a la colina es una técnica de optimización local, el ***Seudocódigo 10*** (tomado de [44]) describe a profundidad el funcionamiento del algoritmo. Este inicia a partir de un individuo aleatorio y realiza una alteración al $\boldsymbol{mejor}$ individuo para así obtener un vecino, si este individuo es mejor que $\boldsymbol{mejor}$ entonces, el mejor individuo es remplazado; este proceso se lleva a cabo hasta alcanzar la máxima cantidad de evaluaciones.

Seudocódigo 10 *Algoritmo Ascenso a la colina.*

Entrada: Máximas evaluaciones $\boldsymbol{maxEvaluaciones} > \mathbf{0}$
Salida: Mejor individuo encontrado $\boldsymbol{mejor}$

```
1:   evaluaciones = 0
2:   mejor = crearIndividuoInicial()
3:   mientras evaluaciones < maxEvaluaciones
4:      vecino = alterar(copia(mejor))
5:      si vecino.evaluacion > mejor.evaluacion entonces
6:         mejor = vecino
7:      fin si
8:   fin mientras
9:   retornar mejor
```

El algoritmo ascenso a la colina, al ser utilizado dentro de la meta-heurística IHDELS, no parte de un individuo aleatorio, en cambio lo hace desde el $\boldsymbol{mejorActual}$ indicado por IHDELS y realiza la evolución desde él.

La alteración del $\boldsymbol{mejor}$ se puede realizar de diversas maneras, en general consiste en cambiar en una pequeña proporción las variables de un individuo; en este trabajo se utilizó el método de **convulsión uniforme limitada** el cual añade ruido a las variables si una probabilidad $\boldsymbol{p}$ se cumple; de ser así se busca un valor aleatorio que al sumarlo con el valor actual de la variable no exceda sus límites permitidos y se le adicionan a la variable [26], dicho procedimiento se explica a detalle en ***Seudocódigo 11*** (tomado de [44]).

Seudocódigo 11 *Alteración: Convulsión Uniforme Limitada.*

Entrada: Solución a alterar $\boldsymbol{s}$,
Probabilidad de añadir ruido a un elemento $\boldsymbol{p}$,
Radio del ruido $\boldsymbol{r}$

Salida: Individuo alterado $\boldsymbol{s}$

```
1:  para i = 0 hasta s.totalVariables
2:     si p ≥ aleatorioDistribuciónUniforme(0, 1) entonces
3:        repetir
4:           n = aleatorio(-r, r)
5:        hasta limiteInferior ≤ s_i + n  ≤ limiteSuperior
6:        s_i = s_i  + n
7:     fin si
8:  fin para
9:  retornar s
```

El ruido que se añade a una variable se escoge de un valor aleatorio entre $[-\boldsymbol{r}, \boldsymbol{r}]$, como se indica en la línea 4 del ***Seudocódigo 11***. Si el valor de $\boldsymbol{r}$ es muy pequeño el algoritmo marchará por un óptimo local y le resultará difícil dar un salto a otra colina, además su velocidad de subida será lenta. Por otro lado si el valor es muy grande, el algoritmo rebotará mucho en el espacio de solución y le será difícil converger cuando se esté acercando al pico de una colina; este valor permite controlar el grado de exploración del algoritmo [26].

3.9.2 Adaptación de SaDE

El algoritmo SaDE es una modificación del algoritmo evolución diferencial, su objetivo es adaptar los valores de la tasa de cruce, y la probabilidad de selección del método de mutación para el siguiente individuo [45], el funcionamiento general corresponde al descrito en el algoritmo DE en el ***Seudocódigo 2***, pero este recibe como parámetros de entrada adicionales otra tasa de cruce $\boldsymbol{CR_2} > \boldsymbol{0}$ y factor de escala $\boldsymbol{F_2} > \boldsymbol{0}$.

Por otro lado, SaDE incluye una variable adicional que es la probabilidad $\boldsymbol{p}$ de seleccionar una estrategia de mutación. SaDE cuenta con dos estrategias de mutación y dependiendo del valor de $\boldsymbol{p}$ se ejecuta una u otra, de esta forma el individuo v resultante en la reproducción se define como se muestra en la ***Ecuación 13***.

Para auto-adaptar la probabilidad $\boldsymbol{p}$ este algoritmo usa el mismo procedimiento que utiliza SaNSDE en la sección **3.7.2**.

El factor de escala $\boldsymbol{F}$ en SaDE se calcula para cada individuo dentro de la reproducción y este valor se obtiene de la ***Ecuación 31***.

$$F_i = N_i(0.5, 0.3)$$

Ecuación 31 *Valor del factor de escala para un individuo.*

Donde $N_i(0.5, 0.3)$ es un valor aleatorio obtenido de una distribución Gaussiana con media 0.5 y desviación estándar 0.3.

Al realizar el cruce se utiliza la ***Ecuación 11*** y el valor de $\boldsymbol{CR}$ es actualizado como se muestra en la ***Ecuación 32*** donde $N_i(CR_m, 0.1)$ es un valor aleatorio con media $\boldsymbol{CR_m}$ y desviación estándar 0.1, cada uno de los valores generados para $\boldsymbol{CR}$ se almacena en un vector llamado $\boldsymbol{CR_{rec}}$, que se utilizará en la ***Ecuación 33*** para calcular el valor de CRm en el siguiente periodo de aprendizaje.

$$CR_i = N_i(CR_m, 0.1)$$

Ecuación 32 *Tasa de cruce para un individuo.*

$$CR_m = \frac{1}{|CR_{rec}|} \sum_{k=1}^{|CR_{rec}|} CR_{rec}(k)$$

Ecuación 33 *Valor de CRm para la siguiente generación.*

Finalmente se realiza el reemplazo de los individuos haciendo uso de la ***Ecuación 12*** comparando todos los individuos de la población con todos los individuos generados en la reproducción y se actualizan las evaluaciones.

Capítulo 4

4. EXPERIMENTACIÓN

El presente capítulo aborda cómo se llevó a cabo el proceso de experimentación sobre el conjunto de datos seleccionado; presentando la comparación con el algoritmo del estado del arte (Memético-ELM). Adicionalmente, incluye un resumen del proceso de afinamiento de parámetros que se realizó a las meta-heurísticas con la idea de obtener mejores resultados de exactitud en el entrenamiento de la red neuronal. Finalmente, se presentan los resultados obtenidos y los análisis estadísticos que se realizaron a los mismos.

En este proyecto se optó por realizar únicamente 3.000 evaluaciones de la función objetivo (EFOs), considerando que una ELM fue concebida para realizar el entrenamiento de una SLFN en menor tiempo que el algoritmo Back Propagation.

La ejecución de los algoritmos meta-heurísticos se realizó en los computadores de las salas de informática de la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca, cuyas características son las siguientes: procesador AMD A-10 a 3.2 GHz, 8 GB RAM y Windows 10.

4.1 AFINAMIENTO DE PARÁMETROS

Las meta-heurísticas DECC-G, MOS e IHDELS fueron sometidas a un proceso de afinamiento de parámetros con el propósito de encontrar el conjunto de valores que mejor se ajusta al problema de entrenamiento de una SLFN con ELM y alcanzar una mejor exactitud en el entrenamiento de la red neuronal.

El afinamiento de parámetros se llevó a cabo utilizando arreglos de cobertura (Covering Arrays, CA). Los CA son estructuras combinatorias representadas matricialmente que formalmente se definen como $CA(N, k, v, t)$, donde N es el número de filas del CA, k representa el número de variables o columnas del CA, v el número de posibles valores (alfabeto o vocabulario) que puede tomar cada variable o columa k y t el nivel de cobertura o fuerza. Se especifica como una matriz de N filas y k columnas de modo que por cada conjunto de t columnas cada t-tupla de v valores sea cubierto en el CA al menos una vez [46].

El número de filas del CA indica el total de pruebas que se deben llevar a cabo para garantizar que todas las posibles combinaciones con una fuerza t sean llevadas a cabo. Por restricciones de tiempo del proyecto la fuerza t utilizada para el afinamiento fue de 2. Los algoritmos meta-heurísticos manejan cada uno un conjunto diferente de parámetros que necesitan ser configurados, por esto el CA de cada algoritmo es diferente; teniendo como característica en común que todos fueron evaluados con $t = 2$ y $v = 5$ y cada configuración fue ejecutada 30 veces, esto último con el propósito de tener valores promedios que respondan a una distribución normal establecido por el teorema del límite central en estadística. Estas pruebas fueron aplicadas para los modelos de entrenamiento-prueba y validación

cruzada, de modo que se obtuvieron diferentes parámetros para cada algoritmo según el tipo de modelo aplicado.

Todas las meta-heurísticas consideradas en este proyecto realizan la evolución trabajando conjuntamente con algoritmos de optimización internos, para conseguir que estos operen adecuadamente el afinamiento de parámetros se realizó de manera conjunta, es decir los parámetros de la meta-heurística y de cualquier otro algoritmo de optimización con el cual opere conjuntamente. Para la selección del conjunto de parámetros en los arreglos de cobertura se utilizó la prueba estadística no paramétrica de Friedman, para realizar la prueba los conjuntos de datos se dividieron en dos subgrupos y no se obtuvo un valor de p válido en ninguna de estas, por lo tanto los resultados no son estadísticamente significativos entre ellos, pero como esto se realizó para encontrar la configuración que obtuviera mejores resultados de manera general, se optó solo por escoger la que menor ranking presentaba sin que fuera propiamente validada por el test de Friedman.

4.1.1 Problemas de clasificación

El afinamiento de parámetros para problemas de clasificación se seleccionó con base en aquellos que tuvieran una cantidad baja, media y alta de atributos y clases. Se tomaron 20 conjuntos de datos para que el proceso no excediera el tiempo de este proyecto, los conjuntos de datos seleccionados fueron: Banknote, Blood, Car, Cardiotocography, Chart, Dermatology, Fertility, Haberman, Hayes, Hill, Ionosphere, Letter, Libras, Optdigits, Pen, QSARBiodegradation, Shuttle, SPECTF, Wilt y Zoo.

4.1.2 Problemas de regresión

Los problemas de regresión seleccionados para este proyecto fueron ocho (8), teniendo en cuenta que la cantidad es baja se decidió realizar el afinamiento utilizando todos los problemas, con el objetivo de tener una cantidad suficiente de datos que permitan tomar una decisión acertada.

4.1.3 Afinamiento de los parámetros de DECC-G

La meta-heurística DECC-G se caracteriza por que en cada ciclo descompone el problema en subcomponentes permitiendo que SaNSDE y DE evolucionen sobre un problema de menor dificultad. Dado que se realizaron 3.000 EFOs era importante garantizar que DECC-G pudiera realizar más de un ciclo, así el problema podría ser descompuesto en varias ocasiones y obtener mejores resultados, con esto en mente, las siguientes variables fueron configuradas como indica la ***Tabla 4*** para así lograr que se llevaran a cabo al menos tres (3) ciclos en DECC-G, esto fue un proceso llevado a cabo meticulosamente, cabe resaltar que SaNSDE y DE también consumen EFOs durante la ejecución de DECC-G y si estas variables no son configuradas adecuadamente puede llegar a suceder que no se complete ni un ciclo, ocasionando que el potencial del algoritmo no sea aprovechado.

Tabla 4. *Variables definidas por defecto en DECC-G*

Variable	Valor	
	Entrenamiento-prueba	**Validación cruzada**
Tamaño población	10	10
Número de subcomponentes	10	6
Evaluaciones SaNSDE (FE)	70	30
Evaluaciones DE (wFE)	100	40

Por otro lado, existe una interoperabilidad entre DECC-G y sus algoritmos internos y dado que estos sí tienen parámetros a los cuales se les pueda realizar el afinamiento se utilizó el $CA(t = 2, k = 6, v = 5)$, en la ***Tabla 5*** se muestra el CA utilizado.

Tabla 5. *CA (2, 6, 5)*

#	(1)	(2)	(3)	(4)	(5)	(6)
1	0	0	2	1	3	0
2	0	1	0	2	4	1
3	0	2	1	4	1	3
4	0	3	3	0	4	0
5	0	4	3	3	0	3
6	1	0	4	3	4	3
7	1	1	1	1	3	2
8	1	2	2	2	2	4
9	1	3	0	1	0	3
10	1	4	3	4	3	1
11	2	0	1	0	0	4
12	2	1	2	0	1	1
13	2	2	3	1	4	1
14	2	3	4	2	3	3
15	2	4	0	3	4	2
16	3	0	0	4	3	4
17	3	1	3	3	2	4
18	3	2	4	0	0	2
19	3	3	2	3	1	3

#	(1)	(2)	(3)	(4)	(5)	(6)
20	3	4	1	2	4	0
21	4	0	3	2	1	2
22	4	1	4	4	0	0
23	4	2	0	0	3	0
24	4	3	1	3	2	1
25	4	4	2	1	4	4
26	0	4	4	1	2	4
27	1	4	4	0	1	3
28	3	*	0	1	1	0
29	2	3	2	4	2	2
30	1	2	*	3	3	0
31	4	0	0	0	2	3
32	3	0	2	2	0	1
33	2	*	*	4	4	0
34	0	*	*	*	*	2
35	*	1	*	*	*	3
36	*	3	*	*	1	4
37	*	*	4	*	*	1
38	*	*	*	*	2	0

Los asteriscos (*) son comodines en el CA, siendo lugares que pueden tomar cualquier valor permitido del vocabulario de su columna respectiva. La ***Tabla 6*** define los valores permitidos para cada parámetro, estos valores son los mismos tanto para validación cruzada (de aquí en adelante se referenciará con la abreviatura *CV*) como para entrenamiento-prueba (de aquí en adelante se referenciará con la abreviatura *TT*).

Tabla 6. *Vocabulario para CA de DECC-G*

Parámetro	Valores				
(1) Tasa de cruce para DE	0.2	0.35	0.5	0.7	0.95
(2) Factor de escala para DE	0.2	0.35	0.5	0.7	0.95
(3) Tasa de cruce 1 para SaNSDE	0.2	0.35	0.5	0.7	0.95
(4) Factor de escala 1 para SaNSDE	0.2	0.35	0.5	0.7	0.95
(5) Tasa de cruce 2 para SaNSDE	0.2	0.35	0.5	0.7	0.95
(6) Factor de escala 2 para SaNSDE	0.2	0.35	0.5	0.7	0.95

Para definir cuál es la mejor configuración de parámetros se aplicó la prueba estadística no paramétrica de Friedman de NxN disponible en KEEL software, sobre los resultados obtenidos. Dado que la matriz obtenida estaba compuesta por 20 filas (conjuntos de datos) y 38 columnas (diferentes configuraciones del algoritmo), se dificultó su procesamiento en la herramienta software, por esto, se decidió dividir el análisis de los resultados de la siguiente manera: los resultados se dividieron en dos partes y se realizó el test de Friedman sobre cada una de las partes por separado, a partir del ranking obtenido de cada una de estas se seleccionaron las 5 mejores configuraciones de cada parte y finalmente se realizó el test de Friedman sobre estos 10 resultados para así finalmente obtener la mejor configuración.

Los resultados de los rankings obtenidos se presentan según cada partición y el ranking obtenido con las 10 mejores configuraciones de cada partición. Los resultados completos del afinamiento de parámetros para DECC-G se encuentran en el **ANEXO I**.

Las tablas ***Tabla 7*** y ***Tabla 8*** muestran los resultados de las particiones realizadas en problemas de clasificación a DECC-G. Las mejores pruebas para cada tipo de problema se resaltan en negrita. De aquí en adelante las tablas presentan cual es el resultado de aplicar el vocabulario al CA según cada prueba, el vocabulario se remplaza según correspondan en el CA, donde el valor 0 representa el parámetro 1, el valor 1 el parámetro 2 y así sucesivamente.

Tabla 7. *Ranking del afinamiento de parámetros parte 1. Problemas de clasificación en DECC-G*

#	Parámetros						Ranking	
	(1)	(2)	(3)	(4)	(5)	(6)	TT	CV
1	0.20	0.20	0.50	0.35	0.70	0.20	11.625	9.675
2	0.20	0.35	0.20	0.50	0.95	0.35	9.5	11.775
3	0.20	0.50	0.35	0.95	0.35	0.70	10.325	11.075
4	0.20	0.70	0.70	0.20	0.95	0.20	**8.65**	9.175
5	0.20	0.95	0.70	0.70	0.20	0.70	9.225	9.275
6	0.35	0.20	0.95	0.70	0.95	0.70	9.6	**8.425**
7	0.35	0.35	0.35	0.35	0.70	0.50	9.15	9.875
8	0.35	0.50	0.50	0.50	0.50	0.95	**8.725**	10.2
9	0.35	0.70	0.20	0.35	0.20	0.70	**8.1**	12.35

10	0.35	0.95	0.70	0.95	0.70	0.35	10.075	**7.025**
11	0.50	0.20	0.35	0.20	0.20	0.95	11.9	**9.15**
12	0.50	0.35	0.50	0.20	0.35	0.35	**9.025**	**9.075**
13	0.50	0.50	0.70	0.35	0.95	0.35	13.05	**8.675**
14	0.50	0.70	0.95	0.50	0.70	0.70	**8.425**	10.675
15	0.50	0.95	0.20	0.70	0.95	0.50	9.225	9.4
16	0.70	0.20	0.20	0.95	0.70	0.95	11.875	11.375
17	0.70	0.35	0.70	0.70	0.50	0.95	9.475	10.875
18	0.70	0.50	0.95	0.20	0.20	0.50	12.925	11.125
19	0.70	0.70	0.50	0.70	0.35	0.70	9.125	10.8

Tabla 8. *Ranking del afinamiento de parámetros parte 2. Problemas de clasificación en DECC-G*

#	Parámetros						Ranking	
	(1)	(2)	(3)	(4)	(5)	(6)	TT	CV
20	0.70	0.95	0.35	0.50	0.95	0.20	10.3	9.25
21	0.95	0.20	0.70	0.50	0.35	0.50	10.625	**8.5**
22	0.95	0.35	0.95	0.95	0.20	0.20	10.025	9.825
23	0.95	0.50	0.20	0.20	0.70	0.20	**8.275**	10.225
24	0.95	0.70	0.35	0.70	0.50	0.35	9.525	10.275
25	0.95	0.95	0.50	0.35	0.95	0.95	9.925	**8.975**
26	0.20	0.95	0.95	0.35	0.50	0.95	**8.8**	12.1
27	0.35	0.95	0.95	0.20	0.35	0.70	**8.85**	11.75
28	0.70	0.35	0.20	0.35	0.35	0.20	**8.8**	11.7
29	0.50	0.70	0.50	0.95	0.50	0.50	10.675	**8.15**
30	0.35	0.50	0.70	0.70	0.70	0.20	9.95	10.05
31	0.95	0.20	0.20	0.20	0.50	0.70	9.525	**8.35**
32	0.70	0.20	0.50	0.50	0.20	0.35	10.35	9.45
33	0.50	0.50	0.35	0.95	0.95	0.20	11.575	9.55
34	0.20	0.95	0.70	0.35	0.20	0.50	9.5	10.825
35	0.35	0.35	0.70	0.95	0.50	0.70	12.35	11.3
36	0.20	0.70	0.50	0.70	0.35	0.95	10.15	**8.325**
37	0.35	0.70	0.95	0.50	0.20	0.35	12.25	10.625
38	0.20	0.35	0.95	0.20	0.50	0.20	**8.55**	10.775

A continuación, la ***Tabla 9*** y la ***Tabla 10*** presenta el ranking final para cada modelo de validación: validación cruzada y entrenamiento-prueba respectivamente con problemas de clasificación.

Tabla 9. *Ranking final de CV. Problemas de clasificación en DECC-G*

#	Parámetros						Ranking
	(1)	(2)	(3)	(4)	(5)	(6)	CV
10	0.35	0.95	0.7	0.95	0.7	0.35	**4.925**
29	0.5	0.7	0.5	0.95	0.5	0.5	4.95
21	0.95	0.2	0.7	0.5	0.35	0.5	5.375
6	0.35	0.2	0.95	0.7	0.95	0.7	5.45
31	0.95	0.2	0.2	0.2	0.5	0.7	5.475
36	0.2	0.7	0.5	0.7	0.35	0.95	5.475
13	0.5	0.5	0.7	0.35	0.95	0.35	5.675
25	0.95	0.95	0.5	0.35	0.95	0.95	5.7
11	0.5	0.2	0.35	0.2	0.2	0.95	5.925
12	0.5	0.35	0.5	0.2	0.35	0.35	6.05

Tabla 10. *Ranking final de TT. Problemas de clasificación en DECC-G*

#	Parámetros						Ranking
	(1)	(2)	(3)	(4)	(5)	(6)	TT
9	0.35	0.7	0.2	0.35	0.2	0.7	**5.1**
38	0.2	0.35	0.95	0.2	0.5	0.2	5.2
14	0.5	0.7	0.95	0.5	0.7	0.7	5.325
26	0.2	0.95	0.95	0.35	0.5	0.95	5.375
23	0.95	0.5	0.2	0.2	0.7	0.2	5.45
4	0.2	0.7	0.7	0.2	0.95	0.2	5.5
12	0.5	0.35	0.5	0.2	0.35	0.35	5.55
8	0.35	0.5	0.5	0.5	0.5	0.95	5.65
27	0.35	0.95	0.95	0.2	0.35	0.7	5.675
28	0.7	0.35	0.2	0.35	0.35	0.2	6.175

Por otro lado, la ***Tabla 11*** y ***Tabla 12*** muestran los resultados de las particiones realizadas en problemas de regresión a DECC-G.

Tabla 11. *Ranking del afinamiento de parámetros parte 1. Problemas de regresión en DECC-G*

#	Parámetros						Ranking	
	(1)	(2)	(3)	(4)	(5)	(6)	TT	CV
1	0.20	0.20	0.50	0.35	0.70	0.20	8.625	**7.75**
2	0.20	0.35	0.20	0.50	0.95	0.35	**8.5625**	12.25
3	0.20	0.50	0.35	0.95	0.35	0.70	11.75	13
4	0.20	0.70	0.70	0.20	0.95	0.20	13.1875	11.625
5	0.20	0.95	0.70	0.70	0.20	0.70	**8.5**	14.0625
6	0.35	0.20	0.95	0.70	0.95	0.70	**4.6875**	10.25
7	0.35	0.35	0.35	0.35	0.70	0.50	11.125	10.125
8	0.35	0.50	0.50	0.50	0.50	0.95	8.625	10.625
9	0.35	0.70	0.20	0.35	0.20	0.70	9.4375	10.4375

10	0.35	0.95	0.70	0.95	0.70	0.35	11.0625	**8.0625**
11	0.50	0.20	0.35	0.20	0.20	0.95	10.6875	**6.875**
12	0.50	0.35	0.50	0.20	0.35	0.35	12.875	8.5625
13	0.50	0.50	0.70	0.35	0.95	0.35	10.0625	8.625
14	0.50	0.70	0.95	0.50	0.70	0.70	**7.4375**	10.625
15	0.50	0.95	0.20	0.70	0.95	0.50	9.6875	**6.4375**
16	0.70	0.20	0.20	0.95	0.70	0.95	12.125	13.875
17	0.70	0.35	0.70	0.70	0.50	0.95	**8.1875**	10
18	0.70	0.50	0.95	0.20	0.20	0.50	9.375	**8**
19	0.70	0.70	0.50	0.70	0.35	0.70	14	8.8125

Tabla 12. *Ranking del afinamiento de parámetros parte 2. Problemas de regresión en DECC-G*

#	Parámetros						Ranking	
	(1)	(2)	(3)	(4)	(5)	(6)	TT	CV
20	0.70	0.95	0.35	0.50	0.95	0.20	10.0625	**7.625**
21	0.95	0.20	0.70	0.50	0.35	0.50	8.625	9.625
22	0.95	0.35	0.95	0.95	0.20	0.20	**7.8125**	**6.625**
23	0.95	0.50	0.20	0.20	0.70	0.20	**8.0625**	12.125
24	0.95	0.70	0.35	0.70	0.50	0.35	9.3125	12
25	0.95	0.95	0.50	0.35	0.95	0.95	8.5625	13.375
26	0.20	0.95	0.95	0.35	0.50	0.95	**6.375**	13
27	0.35	0.95	0.95	0.20	0.35	0.70	11.8125	9.9375
28	0.70	0.35	0.20	0.35	0.35	0.20	10.625	9.125
29	0.50	0.70	0.50	0.95	0.50	0.50	13.6875	**8.5625**
30	0.35	0.50	0.70	0.70	0.70	0.20	10.875	**8.8125**
31	0.95	0.20	0.20	0.20	0.50	0.70	12.0625	8.875
32	0.70	0.20	0.50	0.50	0.20	0.35	8.4375	14.5
33	0.50	0.50	0.35	0.95	0.95	0.20	14.5625	10.3125
34	0.20	0.95	0.70	0.35	0.20	0.50	**7.875**	9.25
35	0.35	0.35	0.70	0.95	0.50	0.70	**7.5625**	11
36	0.20	0.70	0.50	0.70	0.35	0.95	10.875	9.5625
37	0.35	0.70	0.95	0.50	0.20	0.35	12	**5**
38	0.20	0.35	0.95	0.20	0.50	0.20	10.8125	10.6875

Por último, la ***Tabla 13*** y ***Tabla 14*** presenta el ranking final para cada tipo de evaluación: validación cruzada y entrenamiento-prueba respectivamente, con problemas de regresión.

Tabla 13. *Ranking final de CV. Problemas de regresión en DECC-G*

#	Parámetros						Ranking
	(1)	(2)	(3)	(4)	(5)	(6)	CV
37	0.35	0.7	0.95	0.5	0.2	0.35	**4.5**
11	0.5	0.2	0.35	0.2	0.2	0.95	5.0625
15	0.5	0.95	0.2	0.7	0.95	0.5	5.25
22	0.95	0.35	0.95	0.95	0.2	0.2	5.4375

18	0.7	0.5	0.95	0.2	0.2	0.5	5.5
29	0.5	0.7	0.5	0.95	0.5	0.5	5.625
20	0.7	0.95	0.35	0.5	0.95	0.2	5.8125
1	0.2	0.2	0.5	0.35	0.7	0.2	5.8125
10	0.35	0.95	0.7	0.95	0.7	0.35	6
30	0.35	0.5	0.7	0.7	0.7	0.2	6

Tabla 14. *Ranking final de TT. Problemas de regresión en DECC-G*

#	Parámetros						Ranking
	(1)	(2)	(3)	(4)	(5)	(6)	TT
6	0.35	0.2	0.95	0.7	0.95	0.7	**4.25**
26	0.2	0.95	0.95	0.35	0.5	0.95	4.5
34	0.2	0.95	0.7	0.35	0.2	0.5	5.125
35	0.35	0.35	0.7	0.95	0.5	0.7	5.25
14	0.5	0.7	0.95	0.5	0.7	0.7	5.5
17	0.7	0.35	0.7	0.7	0.5	0.95	5.8125
22	0.95	0.35	0.95	0.95	0.2	0.2	5.8125
23	0.95	0.5	0.2	0.2	0.7	0.2	6.0625
2	0.2	0.35	0.2	0.5	0.95	0.35	6.3125
5	0.2	0.95	0.7	0.7	0.2	0.7	6.375

Se observa que la prueba **9** de TT y la prueba **10** de CV para problemas de clasificación y la prueba **6** de TT junto con la prueba **37** de CV para regresión son las que mejor ranking obtienen. De modo que para realizar el entrenamiento de una ELM los parámetros a utilizar serán los que se presentan en la ***Tabla 15***.

Tabla 15. *Parámetros a utilizar en DECC-G*

Parámetro	Clasificación		Regresión	
	TT	CV	TT	CV
(1) Tasa de cruce para DE	0.35	0.35	0.35	0.35
(2) Factor de escala para DE	0.7	0.95	0.2	0.7
(3) Tasa de cruce 1 para SaNSDE	0.2	0.7	0.95	0.95
(4) Factor de escala 1 para SaNSDE	0.35	0.95	0.7	0.5
(5) Tasa de cruce 2 para SaNSDE	0.2	0.7	0.95	0.2
(6) Factor de escala 2 para SaNSDE	0.7	0.35	0.7	0.35

4.1.4 Afinamiento de los parámetros de MOS

Para la meta-heurística MOS se definió el $CA(t = 2, k = 4, v = 5)$**,** su configuración se puede encontrar en la ***Tabla 17***.

El vocabulario utilizado para MOS se especifica en la ***Tabla 16***.

Tabla 16. *Vocabulario para CA de MOS*

Parámetro	Valores				
(1) Número de evaluaciones por iteración (FE)	100	150	200	250	300
(2) Factor de reducción (ξ)	0.15	0.3	0.5	0.7	0.95
(3) Tamaño del vecindario de Solis and Wets	4	8	12	16	20
(4) Rho de Solis and Wets	0.2	0.35	0.5	0.7	0.95

Tabla 17. CA (2, 4, 5)

#	(1)	(2)	(3)	(4)
1	0	0	2	1
2	0	1	0	2
3	0	2	1	4
4	0	3	3	0
5	0	4	3	3
6	1	0	4	3
7	1	1	1	1
8	1	2	2	2
9	1	3	0	1
10	1	4	3	4
11	2	0	1	0
12	2	1	2	0
13	2	2	3	1
14	2	3	4	2
15	2	4	0	3

#	(1)	(2)	(3)	(4)
16	3	0	0	4
17	3	1	3	3
18	3	2	4	0
19	3	3	2	3
20	3	4	1	2
21	4	0	3	2
22	4	1	4	4
23	4	2	0	0
24	4	3	1	3
25	4	4	2	1
26	0	4	4	1
27	1	4	*	0
28	3	*	*	1
29	2	3	2	4
30	*	2	*	3

La ***Tabla 18*** describe los rankings obtenidos como resultados de la ejecución de cada configuración para problemas de clasificación y regresión. Los resultados completos del afinamiento de parámetros de MOS se pueden encontrar en el **ANEXO I.**

Tabla 18. *Ranking del afinamiento de parámetros. Problemas de clasificación y regresión en MOS*

#	Parámetros				Clasificación		Regresión	
	(1)	(2)	(3)	(4)	TT	CV	TT	CV
1	100	0.15	12	0.35	13.575	13.2	17.75	16.5625
2	100	0.3	4	0.5	13.95	**9.475**	15	18.1875
3	100	0.5	8	0.95	11.8	10.45	15.375	10.875
4	100	0.70	16	0.2	13.2	13.05	20	12
5	100	0.95	16	0.7	12.125	14.65	12.125	9.6875
6	150	0.15	20	0.7	15.75	12.325	16.5	14.5625
7	150	0.3	8	0.35	13.325	12.525	15.25	17
8	150	0.50	12	0.5	**9.3**	11.8	19.125	20.25
9	150	0.7	4	0.35	16.725	12.175	14.8125	14.0625
10	150	0.95	16	0.95	14.025	12.175	14.5	13.625
11	200	0.15	8	0.2	16.4	18.5	10.5	20.625

12	200	0.30	12	0.2	15.925	15.225	19.5	15.8125
13	200	0.50	16	0.35	13.2	16.375	15.4375	15.3125
14	200	0.70	20	0.5	18.05	19.575	13.9375	17.6875
15	200	0.95	4	0.7	17.175	19.85	14	18.5625
16	250	0.15	4	0.95	15.875	20.425	**8.625**	20.1875
17	250	0.30	16	0.7	18.325	19	10.625	16.1875
18	250	0.50	20	0.2	17.9	16.275	17.875	16.125
19	250	0.70	12	0.7	18.1	17.975	14.875	15.25
20	250	0.95	8	0.5	17.8	20.275	12.5	16.1875
21	300	0.15	16	0.5	16.3	16.275	14.375	17.875
22	300	0.30	20	0.95	16.15	16.225	19	17.875
23	300	0.5	4	0.2	17.525	19.1	17.25	**8.4375**
24	300	0.7	8	0.7	17.35	19.725	15.875	8.9375
25	300	0.95	12	0.35	21.55	18.9	14.625	19.625
26	100	0.95	20	0.35	15.65	12.95	20.375	13.375
27	150	0.95	16	0.2	14.575	12.2	17.8125	14.5625
28	250	0.95	12	0.35	17.3	18.225	14.9375	15.25
29	200	0.70	12	0.95	13.675	15.225	13.6875	17.3125
30	100	0.5	8	0.7	12.4	10.875	18.75	13

Aquí la mejor configuración para problemas de clasificación se obtuvo con la prueba **8** para TT y la prueba **2** para CV; para los problemas de regresión las mejores pruebas fueron la **16** para TT y la **23** para CV. A partir de esto se determina que los valores a usar para los parámetros de MOS son los presentados en la ***Tabla 19***.

Tabla 19. *Parámetros a utilizar en MOS*

Parámetro	**Clasificación**		**Regresión**	
	TT	**CV**	**TT**	**CV**
(1) Número de evaluaciones por iteración (FE)	150	100	250	300
(2) Factor de reducción (ξ)	0.5	0.3	0.15	0.5
(3) Tamaño del vecindario de Solis and Wets	12	4	4	4
(4) Rho de Solis and Wets	0.5	0.5	0.95	0.2

4.1.5 Afinamiento de los parámetros de IHDELS

Por último, la meta-heurística IHDELS fue afinada con el $CA(t = 2, k = 12, v = 5)$**,** la representación de este CA se describe en la

Tabla 20.

Tabla 20. *CA (2, 12, 5)*

#	**(1)**	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**	**(8)**	**(9)**	**(10)**	**(11)**	**(12)**
1	0	0	2	1	3	0	2	4	4	3	3	3
2	0	1	0	2	4	1	3	4	4	1	0	2
3	0	2	1	4	1	3	4	0	2	0	0	2

4	0	3	3	0	4	0	0	0	1	0	4	0
5	0	4	3	3	0	3	1	4	3	2	1	2
6	1	0	4	3	4	3	1	1	2	2	0	0
7	1	1	1	1	3	2	0	3	1	2	2	1
8	1	2	2	2	2	4	1	3	2	3	4	1
9	1	3	0	1	0	3	4	1	0	4	2	3
10	1	4	3	4	3	1	3	2	4	4	4	4
11	2	0	1	0	0	4	1	2	1	1	0	4
12	2	1	2	0	1	1	4	1	3	3	4	4
13	2	2	3	1	4	1	0	2	2	4	1	3
14	2	3	4	2	3	3	4	0	1	0	3	2
15	2	4	0	3	4	2	2	3	2	0	0	4
16	3	0	0	4	3	4	0	1	3	0	1	1
17	3	1	3	3	2	4	4	4	2	1	2	4
18	3	2	4	0	0	2	2	1	4	1	4	2
19	3	3	2	3	1	3	3	2	1	3	0	0
20	3	4	1	2	4	0	3	1	3	4	3	0
21	4	0	3	2	1	2	2	2	3	4	0	1
22	4	1	4	4	0	0	1	3	2	4	3	3
23	4	2	0	0	3	0	4	2	2	2	3	4
24	4	3	1	3	2	1	2	1	1	4	3	4
25	4	4	2	1	4	4	4	4	4	0	2	0
26	0	4	4	1	2	4	3	0	3	4	2	4
27	1	4	4	0	1	3	0	4	3	1	3	1
28	3	4	0	1	1	0	1	0	0	1	0	3
29	2	3	2	4	2	2	3	4	1	2	1	3
30	1	2	1	3	3	0	3	4	0	0	4	3
31	4	0	0	0	2	3	3	3	0	4	4	2
32	3	0	2	2	0	1	0	0	0	2	2	2
33	2	1	4	4	4	0	2	2	0	3	2	0
34	0	0	3	*	0	2	4	3	0	3	3	1
35	1	1	1	2	1	3	2	0	4	0	1	4
36	3	3	*	0	1	4	1	3	4	2	2	3
37	4	3	4	4	3	1	1	3	3	1	1	0
38	4	*	1	3	2	0	0	0	4	3	1	1
39	0	3	3	0	0	1	3	1	2	0	1	1
40	2	*	*	1	0	2	1	0	4	0	4	0
41	0	*	*	1	2	4	2	2	0	0	3	2
42	*	4	0	*	2	*	0	*	1	3	0	0
43	*	2	2	*	*	*	*	*	1	1	2	0
44	*	2	2	2	3	*	*	4	3	4	0	3
45	2	*	*	*	4	*	2	*	*	2	4	1
46	*	*	*	*	*	*	4	*	0	*	1	4
47	1	*	*	*	*	0	*	*	*	3	*	2
48	*	*	*	*	*	*	0	*	*	*	*	4

El vocabulario definido para esta meta-heurística se presenta en la ***Tabla 21***.

Tabla 21. *Vocabulario par CA de IHDELS*

Parámetro	Valores				
(1) Número de evaluaciones a dar en DE en IHDELS (FE_DE)	30	45	60	75	90
(2) Número de evaluaciones a dar para una BL (FE_LS)	30	45	60	75	90
(3) Límite inferior del dominio de búsqueda (a)	-1	-0.8	-0.6	-0.4	-0.2
(4) Límite superior del dominio de búsqueda (b)	1	0.8	0.6	0.4	0.2
(5) Número de reinicios	2	3	5	8	12
(6) Valor umbral	0.01	0.05	0.11	0.2	0.4
(7) Tasa de cruce 1 de SaDE	0.2	0.35	0.5	0.7	0.95
(8) Factor de escala 1 de SaDE	0.2	0.35	0.5	0.7	0.95
(9) Tasa de cruce 2 de SaDE	0.2	0.35	0.5	0.7	0.95
(10) Factor de escala 2 de SaDE	0.2	0.35	0.5	0.7	0.95
(11) Probabilidad de Ascenso por la colina	0.2	0.35	0.5	0.7	0.95
(12) Radio del ruido de ascenso por la colina	0.1	0.2	0.35	0.6	0.85

La ***Tabla 22*** presenta únicamente el resultado de aplicar el vocabulario al CA haciéndose así por razones de espacio.

Tabla 22. *Vocabulario aplicado al CA de IHDELS*

#	Parámetros											
	(1)	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**	**(8)**	**(9)**	**(10)**	**(11)**	**(12)**
1	30	30	-0.60	0.80	8	0.01	0.50	0.95	0.95	0.70	0.70	0.6
2	30	45	-1	0.60	12	0.05	0.70	0.95	0.95	0.35	0.20	0.35
3	30	60	-0.80	0.20	3	0.20	0.95	0.20	0.50	0.20	0.20	0.35
4	30	75	-0.40	1	12	0.01	0.20	0.20	0.35	0.20	0.95	0.1
5	30	90	-0.40	0.40	2	0.20	0.35	0.95	0.70	0.50	0.35	0.35
6	45	30	-0.20	0.40	12	0.20	0.35	0.35	0.50	0.50	0.20	0.1
7	45	45	-0.80	0.80	8	0.10	0.20	0.70	0.35	0.50	0.50	0.2
8	45	60	-0.60	0.60	5	0.40	0.35	0.70	0.50	0.70	0.95	0.2
9	45	75	-1	0.80	2	0.20	0.95	0.35	0.20	0.95	0.50	0.6
10	45	90	-0.40	0.20	8	0.05	0.70	0.50	0.95	0.95	0.95	0.85
11	60	30	-0.80	1	2	0.40	0.35	0.50	0.35	0.35	0.20	0.85
12	60	45	-0.60	1	3	0.05	0.95	0.35	0.70	0.70	0.95	0.85
13	60	60	-0.40	0.80	12	0.05	0.20	0.50	0.50	0.95	0.35	0.6
14	60	75	-0.20	0.60	8	0.20	0.95	0.20	0.35	0.20	0.70	0.35
15	60	90	-1	0.40	12	0.10	0.50	0.70	0.50	0.20	0.20	0.85
16	75	30	-1	0.20	8	0.40	0.20	0.35	0.70	0.20	0.35	0.2
17	75	45	-0.40	0.40	5	0.40	0.95	0.95	0.50	0.35	0.50	0.85
18	75	60	-0.20	1	2	0.10	0.50	0.35	0.95	0.35	0.95	0.35

19	75	75	-0.60	0.40	3	0.20	0.70	0.50	0.35	0.70	0.20	0.1
20	75	90	-0.80	0.60	12	0.01	0.70	0.35	0.70	0.95	0.70	0.1
21	90	30	-0.40	0.60	3	0.10	0.50	0.50	0.70	0.95	0.20	0.2
22	90	45	-0.20	0.20	2	0.01	0.35	0.70	0.50	0.95	0.70	0.6
23	90	60	-1	1	8	0.01	0.95	0.50	0.50	0.50	0.70	0.85
24	90	75	-0.80	0.40	5	0.05	0.50	0.35	0.35	0.95	0.70	0.85
25	90	90	-0.60	0.80	12	0.40	0.95	0.95	0.95	0.20	0.50	0.1
26	30	90	-0.20	0.80	5	0.40	0.70	0.20	0.70	0.95	0.50	0.85
27	45	90	-0.20	1	3	0.20	0.20	0.95	0.70	0.35	0.70	0.2
28	75	90	-1	0.80	3	0.01	0.35	0.20	0.20	0.35	0.20	0.6
29	60	75	-0.60	0.20	5	0.10	0.70	0.95	0.35	0.50	0.35	0.6
30	45	60	-0.80	0.40	8	0.01	0.70	0.95	0.20	0.20	0.95	0.6
31	90	30	-1	1	5	0.20	0.70	0.70	0.20	0.95	0.95	0.35
32	75	30	-0.60	0.60	2	0.05	0.20	0.20	0.20	0.50	0.50	0.35
33	60	45	-0.20	0.20	12	0.01	0.50	0.50	0.20	0.70	0.50	0.1
34	30	30	-0.40	0.60	2	0.10	0.95	0.70	0.20	0.70	0.70	0.2
35	45	45	-0.80	0.60	3	0.20	0.50	0.20	0.95	0.20	0.35	0.85
36	75	75	-0.60	1	3	0.40	0.35	0.70	0.95	0.50	0.50	0.6
37	90	75	-0.20	0.20	8	0.05	0.35	0.70	0.70	0.35	0.35	0.1
38	90	60	-0.80	0.40	5	0.01	0.20	0.20	0.95	0.70	0.35	0.2
39	30	75	-0.40	1	2	0.05	0.70	0.35	0.50	0.20	0.35	0.2
40	60	45	-1	0.80	2	0.10	0.35	0.20	0.95	0.20	0.95	0.1
41	30	75	-0.60	0.80	5	0.40	0.50	0.50	0.20	0.20	0.70	0.35
42	45	90	-1	0.60	5	0.20	0.20	0.95	0.35	0.70	0.20	0.1
43	45	60	-0.60	0.40	2	0.10	0.35	0.95	0.35	0.35	0.50	0.1
44	90	60	-0.60	0.60	8	0.40	0.50	0.95	0.70	0.95	0.20	0.6
45	60	30	-0.80	0.60	12	0.20	0.50	0.20	0.35	0.50	0.95	0.2
46	45	60	-0.40	1	12	0.01	0.95	0.35	0.20	0.50	0.35	0.85
47	45	90	-0.60	0.80	2	0.01	0.70	0.35	0.95	0.70	0.95	0.35
48	75	45	-0.60	0.80	2	0.40	0.20	0.70	0.50	0.20	0.35	0.85

Dada la complejidad de procesamiento por el tamaño del CA para obtener el ranking a partir de las pruebas de Friedman, se decidió dividir los resultados en dos tablas separadas, siguiendo el mismo procedimiento que se explicó en la sección **4.1.3**. Los resultados del afinamiento de parámetros para IHDELS se encuentran en el **ANEXO I**. Con el objetivo de no extender la sección, únicamente se presentan los rankings finales.

La ***Tabla 23*** y ***Tabla 24*** presenta el ranking final para cada modelo: validación cruzada y evaluación y pruebas respectivamente con problemas de clasificación.

Tabla 23. *Ranking final de CV. Problemas de clasificación en IHDELS*

#	Ranking CV
33	**4.575**
15	4.8
5	4.95
21	5.15
25	5.425
31	5.475
40	5.875
37	5.95
24	6.3
4	6.5

Tabla 24. *Ranking final de TT. Problemas de clasificación en IHDELS*

#	Ranking TT
44	**3.9**
38	4.725
24	4.9
14	5
23	5.6
4	5.875
15	5.925
37	6.15
39	6.3
45	6.625

La ***Tabla 25*** y ***Tabla 26*** presenta el ranking consolidado para cada modelo de validación con problemas de regresión.

Tabla 25. *Ranking final de CV. Problemas de regresión en IHDELS*

#	Ranking CV
26	**4.8125**
37	4.875
46	4.9375
39	5.0625
14	5.375
3	5.4375
12	5.6875
20	5.875
5	6.25
28	6.6875

Tabla 26. *Ranking final de TT. Problemas de regresión en IHDELS*

#	Ranking
	TT
28	**4.75**
2	4.8125
13	4.875
45	4.9375
16	5.625
38	5.8125
19	5.875
24	5.9375
41	6
30	6.375

Los resultados obtenidos muestran que las configuraciones más adecuadas para problemas de clasificación son la **44** y **33**, y para problemas de regresión la **28** y **26** en los modelos de validación TT y CV respectivamente. La ***Tabla 27*** presenta el resumen final de los parámetros a utilizar.

Tabla 27. *Parámetros a utilizar en IHDELS*

Parámetro	Clasificación		Regresión	
	TT	**CV**	**TT**	**CV**
(1) Número de evaluaciones a dar a DE en IHDELS (FE_DE)	90	60	75	30
(2) Número de evaluaciones a dar para una BL (FE_LS)	60	45	90	90
(3) Límite inferior del dominio de búsqueda (a)	-0.60	-0.20	-1	-0.20
(4) Límite superior del dominio de búsqueda (b)	0.60	0.20	0.8	0.80
(5) Número de reinicios	8	12	3	5
(6) Valor umbral	0.40	0.01	0.01	0.40
(7) Tasa de cruce 1 de SaDE	0.50	0.50	0.35	0.70
(8) Factor de escala 1 de SaDE	0.95	0.50	0.2	0.20
(9) Tasa de cruce 2 de SaDE	0.70	0.20	0.2	0.70
(10) Factor de escala 2 de SaDE	0.95	0.70	0.35	0.95
(11) Probabilidad de Ascenso por la colina	0.20	0.50	0.2	0.50
(12) Radio del ruido de Ascenso por la colina	0.6	0.1	0.6	0.85

4.1.6 Afinamiento de los parámetros de M-ELM

De acuerdo a lo planteado en [4] los parámetros a utilizar son los especificados en la ***Tabla 28***; puesto que en [4] no se manifiesta una diferenciación de los valores según el modelo de validación o de tipo de problema, se asume que se utiliza el mismo en los dos casos.

Tabla 28 *Parámetros a utilizar en M-ELM*

Parámetro	Clasificación		Regresión	
	TT	CV	TT	CV
(1) Tamaño de la población	50	50	50	50
(2) Cantidad de optimizaciones locales	70	70	70	70

4.2 RESULTADOS

Esta sección presenta los resultados obtenidos de ejecutar cada uno de los problemas de clasificación y regresión con los algoritmos meta-heurísticos implementados, los resultados se obtuvieron de realizar 3.000 evaluaciones de la función objetivo. Cada algoritmo fue ejecutado con los problemas de regresión y clasificación así mismo como con los modelos de validación entrenamiento-prueba y validación cruzada. Así, los resultados se dividieron en cuatro secciones, de los cuales se presentan los valores promedio obtenidos de ejecutar treinta (30) veces un algoritmo con un conjunto de datos y un modelo de validación; cada ejecución se hizo con una semilla diferente.

Adicionalmente también se definió que:

- Los problemas de validación cruzada tendrían diez (10) subconjuntos o folders, es decir $k = 10$.
- La ELM utiliza 50 neuronas en la capa oculta para el entrenamiento de la red neuronal, para hacer una comparación justa con el trabajo M-ELM [4].
- La función de activación que se utilizó en la capa oculta es la función Sigmoidal.

4.2.1 Comparación con M-ELM y caminata aleatoria

Los parámetros utilizados para ejecutar DECC-G, MOS, IHDELS y M-ELM se muestran en la ***Tabla 15***, ***Tabla 19***, ***Tabla 27*** y ***Tabla 28*** respectivamente. También se obtuvieron resultados de experimentación sobre el algoritmo de caminata aleatoria (denominado en este trabajo como: random) para obtener también una comparación justa contra el funcionamiento original de la ELM.

Los resultados detallan la exactitud obtenida según el tipo de problema evaluado que corresponden al promedio y la desviación estándar de las 30 ejecuciones del algoritmo de la columna con el conjunto de datos de la fila. El algoritmo con mejor resultado en cada conjunto de datos se presenta en negrita.

La ***Tabla 29*** ilustra los resultados obtenidos para problemas de clasificación con el modelo de entrenamiento-prueba.

Tabla 29. *Comparación de entrenamiento-prueba en clasificación*

Conjunto de datos	1. DECC-G	2. MOS	3. IHDELS	4. M-ELM	5. Random
Banknote	0.9999±0.0004	0.9998±0.0007	**1±0**	0.9999±0.0006	0.9998±0.0007
Blood	**0.754±0.0094**	0.7528±0.0097	0.7508±0.0092	0.7491±0.0068	0.7527±0.0106
Car	0.8398±0.0188	**0.8466±0.025**	0.8465±0.0218	0.8411±0.0224	0.8399±0.0227
Cardiotocography	0.7678±0.01	0.7623±0.0112	**0.7742±0.0094**	0.7675±0.0092	0.7671±0.011
Chart	0.9063±0.0193	**0.917±0.019**	0.917±0.0233	0.9078±0.0218	0.9092±0.0228
Climatesimulation	0.8491±0.0033	0.8472±0.0012	**0.8534±0.009**	0.8494±0.0039	0.8515±0.0053
Connectionist	**0.7629±0.0465**	0.76±0.0461	0.7567±0.0395	0.7529±0.0501	0.7505±0.0431
Contraceptive	0.5276±0.0156	0.5307±0.0126	**0.5345±0.0123**	0.5298±0.0134	0.527±0.0135
Dermatology	**0.971±0.0121**	0.9672±0.0144	0.9648±0.011	0.9678±0.0106	0.9678±0.0108
Diabetes	**0.7578±0.0128**	0.7535±0.0149	0.7531±0.0128	0.7569±0.0165	0.7509±0.0119
Ecoli	**0.8568±0.0152**	0.8494±0.0167	0.8568±0.0169	0.847±0.0138	0.8488±0.0126
Fertility	**0.7111±0.0395**	0.6722±0.0626	0.7033±0.0697	0.6556±0.0717	0.6644±0.0655
Glass	0.5224±0.0422	**0.531±0.0409**	0.5248±0.0503	0.5229±0.0542	0.5252±0.0422
Haberman	0.7036±0.0172	**0.7101±0.0138**	0.7026±0.02	0.7098±0.014	0.7029±0.0146
Hayes	0.6826±0.0545	0.6712±0.0516	**0.6864±0.0555**	0.6515±0.0646	0.6553±0.0557
Hill	0.6655±0.0146	0.6518±0.0198	0.6587±0.0223	0.6556±0.02	**0.6696±0.0263**
Indian	**0.6444±0.0157**	0.6439±0.016	0.644±0.0166	0.6416±0.0155	0.6432±0.015
Ionosphere	**0.8635±0.0264**	0.8561±0.0274	0.8493±0.0282	0.8456±0.0286	0.8476±0.0341
Iris	0.8953±0.0406	0.8853±0.039	**0.9127±0.0424**	0.8716±0.0406	0.876±0.0312
Knowledge	0.8256±0.0269	**0.8298±0.0211**	0.8236±0.0257	0.8271±0.0234	0.8279±0.0246
Leaf	0.7767±0.03	0.7758±0.0245	**0.783±0.0192**	0.7715±0.0215	0.7709±0.02
Letter	**0.7056±0.004**	0.6898±0.0057	0.7011±0.0062	0.6879±0.0052	0.6947±0.0057
Libras	**0.7777±0.0334**	0.7713±0.0434	0.769±0.034	0.7597±0.0308	0.769±0.0372
Optdigits	0.8911±0.0117	0.8823±0.0099	**0.8965±0.0168**	0.8868±0.0131	0.8858±0.0132
Pen	**0.9305±0.0076**	0.9163±0.0078	0.9229±0.0084	0.9152±0.009	0.9191±0.0086
Planning	0.5196±0.0437	0.5174±0.0518	0.5256±0.044	**0.5365±0.0445**	0.5146±0.0529
QSARBiodegradation	0.8608±0.0125	0.8612±0.012	**0.8622±0.0094**	0.8564±0.0107	0.8614±0.0111
Seeds	**0.981±0.01**	0.9776±0.0126	0.9805±0.0101	0.9757±0.0153	0.979±0.0164
Shuttle	0.5059±0.2992	**0.5626±0.2804**	0.504±0.2991	0.5303±0.2399	0.4587±0.2781
SPECTF	0.3213±0.1378	**0.3408±0.129**	0.3012±0.1209	0.3202±0.1455	0.325±0.1114
Vertebral2C	0.8597±0.0152	0.8537±0.0204	**0.867±0.0197**	0.8573±0.0197	0.8537±0.0196
Vertebral3C	0.8133±0.0176	0.816±0.0227	**0.817±0.0159**	0.8133±0.0162	0.8147±0.0182
Wdbc	0.9625±0.0078	0.9607±0.0089	**0.9632±0.0083**	0.9621±0.0091	0.96±0.0108
Wilt	0.9694±0.0016	0.9689±0.0018	0.97±0.0014	0.969±0.0017	**0.9704±0.0016**
Wine	**0.9761±0.017**	0.955±0.0276	0.9722±0.0203	0.9611±0.0199	0.9617±0.0232
WineRed	**0.5891±0.0134**	0.5846±0.0136	0.5889±0.011	0.5875±0.0105	0.5866±0.014
Yeast	**0.5792±0.0067**	0.5764±0.0084	0.5752±0.0081	0.5764±0.0056	0.5782±0.008
Zoo	**0.8796±0.028**	0.8527±0.0376	0.8602±0.0372	0.8108±0.0563	0.8065±0.0507

A continuación, la ***Tabla 30*** expone los resultados para problemas de clasificación con el modelo de validación cruzada.

Tabla 30. *Comparación de validación cruzada en clasificación*

Conjunto de datos	6. DECC-G	7. MOS	8. IHDELS	9. M-ELM	10. Random
Banknote	**0.9999±0.0004**	0.9998±0.0007	**0.9999±0.0004**	0.9998±0.0007	0.9998±0.0007
Blood	**0.7504±0.0072**	0.7468±0.0088	0.7485±0.0091	0.747±0.01	0.7484±0.0085
Car	0.836±0.0215	0.8425±0.0203	0.8366±0.0226	**0.8456±0.0192**	0.8436±0.0204
Cardiotocography	0.765±0.0086	0.7617±0.0097	0.7678±0.0122	**0.7682±0.0099**	0.7651±0.0117
Chart	0.9117±0.023	0.9057±0.0252	**0.9188±0.0188**	0.9095±0.0205	0.916±0.0188
Climatesimulation	0.8492±0.0046	0.8478±0.0016	0.8495±0.0054	0.8489±0.003	**0.8503±0.005**
Connectionist	**0.7662±0.0466**	0.7371±0.048	0.7367±0.055	0.7429±0.0394	0.7576±0.0486
Contraceptive	0.5275±0.0114	0.5287±0.0148	**0.534±0.0125**	0.5297±0.0135	0.5289±0.0141
Dermatology	0.9615±0.0151	0.9661±0.0123	0.9612±0.0125	**0.9693±0.0102**	0.9642±0.0093
Diabetes	0.7549±0.0116	0.7542±0.0187	0.7543±0.0125	**0.7575±0.0151**	0.7569±0.0154
Ecoli	0.8491±0.0173	**0.8521±0.013**	0.8488±0.0163	0.8506±0.0103	0.8488±0.0151
Fertility	0.6567±0.0717	0.6433±0.0588	**0.6689±0.0711**	0.6448±0.072	0.66±0.0674

Glass	0.549±0.0357	**0.5552±0.0438**	0.5305±0.0439	0.535±0.0314	0.5419±0.0492
Haberman	**0.7265±0.0144**	0.7261±0.0174	0.7258±0.017	0.723±0.0186	0.7222±0.0163
Hayes	**0.678±0.0563**	0.6561±0.045	0.675±0.0646	0.6712±0.0465	0.6583±0.0517
Hill	0.6384±0.0278	0.6383±0.0252	**0.6564±0.0249**	0.6467±0.0218	0.6459±0.0232
Indian	0.6405±0.0178	0.6381±0.0167	0.6386±0.0167	**0.6423±0.016**	0.6358±0.0183
Ionosphere	0.8493±0.0298	**0.851±0.0257**	0.841±0.0265	0.8459±0.0252	0.8413±0.0292
Iris	0.8827±0.0409	**0.892±0.0409**	0.892±0.0431	0.8677±0.051	0.878±0.0394
Knowledge	**0.8461±0.0234**	0.8314±0.0253	0.8434±0.0236	0.8408±0.0213	0.838±0.0225
Leaf	0.7745±0.0194	0.7721±0.0257	0.7755±0.0195	**0.78±0.0251**	0.7779±0.0217
Letter	0.6887±0.0055	0.6859±0.0058	**0.6917±0.0065**	0.6903±0.0042	0.6887±0.0054
Libras	0.7783±0.0323	0.7697±0.033	**0.7843±0.0346**	0.7667±0.0397	0.7767±0.0355
Optdigits	0.8835±0.012	0.8767±0.0121	**0.8886±0.0142**	0.8883±0.0142	0.8825±0.0114
Pen	0.9139±0.011	0.9057±0.0101	0.9153±0.01	**0.9155±0.0075**	0.9152±0.0102
Planning	0.5306±0.0424	**0.5402±0.0425**	0.5224±0.0443	0.5292±0.0502	0.5242±0.0316
QSARBiodegradation	0.8613±0.0115	0.8608±0.0115	**0.8629±0.0118**	0.8582±0.0095	0.8551±0.0077
Seeds	**0.9733±0.0109**	0.9695±0.0183	0.9705±0.0169	0.9705±0.0195	0.9714±0.0143
Shuttle	0.4123±0.268	**0.5787±0.2897**	0.5082±0.3006	0.5261±0.2757	0.439±0.2886
SPECTF	**0.3517±0.1419**	0.3513±0.1584	0.3471±0.1179	0.3267±0.1076	0.3387±0.1193
Vertebral2C	0.858±0.0202	0.8597±0.0156	0.8603±0.0192	0.8643±0.0193	**0.869±0.0209**
Vertebral3C	0.8057±0.018	0.8043±0.0191	0.8037±0.0178	**0.8137±0.0168**	0.8067±0.0194
Wdbc	0.9582±0.0081	0.9618±0.0079	0.9611±0.011	**0.9619±0.0082**	0.9591±0.0106
Wilt	0.9687±0.0016	0.9687±0.0015	0.9695±0.0013	0.9694±0.0017	**0.9699±0.0014**
Wine	0.9606±0.023	0.9628±0.0218	0.9611±0.0273	0.9517±0.0217	**0.9639±0.0244**
WineRed	0.5849±0.0115	0.5816±0.0101	0.5851±0.0088	**0.5867±0.0078**	0.5861±0.0091
Yeast	0.5756±0.0068	**0.578±0.0073**	0.5724±0.0065	0.5734±0.0097	0.5733±0.008
Zoo	0.7462±0.1568	0.7387±0.133	0.6763±0.1927	**0.7634±0.1422**	0.7441±0.1213

Por último, la ***Tabla 31*** y ***Tabla 32*** contienen los resultados en problemas de regresión con el modelo de entrenamiento-prueba y validación cruzada respectivamente. Es importante recordar que los resultados aquí presentados corresponden a la exactitud obtenida de la ELM para problemas de regresión definida en la ***Ecuación 7***.

Tabla 31. *Comparación de entrenamiento-prueba en regresión*

Conjunto de datos	1. DECC-G	2. MOS	3. IHDELS	4. M-ELM	5. Random
AutoMpg	0.218±0.0071	**0.2247±0.0068**	0,2221±0,0072	0.2225±0.0062	0.2211±0.0071
AutoPrice	0.0002±0.0001	0.0003±0.0001	**0,0003±0**	**0.0003±0**	**0.0003±0**
BodyFat	0.4381±0.0641	0.4354±0.0768	**0,4403±0,0673**	0.4292±0.0592	0.4329±0.0693
Cpu	0.028±0.0252	**0.04±0.0373**	0,0349±0,0272	0.031±0.0241	0.0382±0.0341
Housing	**0.1978±0.015**	0.1866±0.0118	0,1886±0,0119	0.1886±0.0105	0.1929±0.0114
Sensory	**0.5534±0.0079**	0.5524±0.0093	0,5523±0,0081	0.5519±0.009	0.5523±0.0048
Servo	0.4662±0.015	0.4674±0.0166	**0,4688±0,0206**	0.4669±0.021	0.469±0.0162
Veteran	0.0033±0.0011	**0.0041±0.001**	0,0041±0,0011	0.0041±0.0011	0.0038±0.0013

Tabla 32. *Comparación de validación cruzada en regresión*

Conjunto de datos	6. DECC-G	7. MOS	8. IHDELS	9. M-ELM	10. Random
AutoMpg	0.2217±0.0067	0.222±0.007	**0.224±0.0079**	0.2199±0.0065	0.2189±0.0057
AutoPrice	0.0003±0.0001	**0.0003±0**	0.0003±0.0001	0.0003±0.0001	0.0003±0.0001
BodyFat	0.4326±0.0546	0.4219±0.0595	0.4211±0.0649	**0.4331±0.0557**	0.4249±0.076
Cpu	0.0425±0.0306	**0.0511±0.0496**	0.0341±0.032	0.0385±0.0371	0.0459±0.0364
Housing	0.1896±0.0115	0.1901±0.0134	0.1905±0.0108	**0.1935±0.0104**	0.1906±0.0105
Sensory	**0.5526±0.0079**	0.5466±0.0081	0.5495±0.0084	0.5516±0.0066	0.5508±0.0071
Servo	0.4656±0.016	0.4687±0.0121	**0.477±0.0143**	0.4662±0.0191	0.4639±0.0198
Veteran	0.004±0.0011	0.0044±0.0013	0.004±0.001	**0.0044±0.0009**	0.0041±0.001

4.2.2 Análisis estadístico

Un análisis estadístico fue llevado a cabo con las pruebas estadísticas no paramétricas de Wilcoxon y Friedman con el objetivo de establecer el algoritmo meta-heurístico que presenta un mejor desempeño.

La prueba de Wilcoxon establece un nivel de dominancia de los resultados de un algoritmo sobre otro. El punto negro indica que el algoritmo de la fila domina al algoritmo de la columna y el punto blanco indica que el algoritmo de la columna domina al de la fila, la casilla vacía indica que no es posible establecer una dominancia de un algoritmo sobre otro y los resultados por debajo de la diagonal tienen una significancia de 0.95 y por encima de la diagonal de 0.90.

La ***Tabla 33*** presenta los resultados obtenidos de la prueba de Wilcoxon para problemas de clasificación donde se evidencia que el algoritmo con mayor dominancia es IHDELS con el modelo de validación entrenamiento-prueba (TT) con una significancia del 0.95; IHDELS TT domina a todos los demás algoritmos a excepción de DECC-G TT quien es el segundo en presentar mayor dominancia, seguido por MOS TT.

Tabla 33 *Resultados de la prueba de Wilcoxon en problemas de clasificación*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
DECC-G TT (1)	-		•	•	•	•	•	•	•	•
IHDELS TT (2)		-	•	•	•	•	•	•	•	•
MOS TT (3)	○	○	-	•	•					
M-ELM TT (4)	○	○	○	-						
RW TT (5)	○	○	○		-					
DECC-G CV (6)	○	○				-				
IHDELS CV (7)	○	○					-			
MOS CV (8)		○						-		
M-ELM CV(9)	○	○							-	
RW CV (10)	○	○								-

En problemas de regresión no se pudo establecer con base en el test de Wilcoxon ninguna dominancia entre los algoritmos.

Además, al aplicar la prueba de Friedman la cual establece un ranking entre los algoritmos ratifica que el algoritmo ganador es IHDELS TT para problemas de clasificación usando el modelo de entrenamiento-prueba, seguido de DECC-G TT, como se evidencia en la ***Tabla 34***, el valor p obtenido fue igual a 5.80E-6 siendo menor a 0.05 y distribuido con un chi cuadrado con 9 grados de libertad igual a

40.646, lo que significa que estadísticamente los resultados son representativos. Estos dos superan notablemente al resto de los algoritmos con su correspondiente modelo de validación.

Tabla 34. *Prueba de Friedman en problemas de clasificación*

Algoritmo	Ranking
IHDELS TT	**3.5526**
DECC-G TT	3.8026
MOS TT	5.3421
IHDELS CV	5.5395
M-ELM CV	5.5658
Random TT	5.9605
DECC-G CV	6
Random CV	6.1974
M-ELM TT	6.3289
MOS CV	6.7105

Teniendo en cuenta que los resultados obtenidos para problemas de clasificación obtuvieron un valor de P menor a 0.05 se realiza un análisis post-hoc para determinar si las diferencias entre los ranking son significativamente diferentes, aplicando la prueba de Nemenyi [47]. A continuación, se presenta el valor de la distancia crítica en la ***Ecuación 34***.

$$CD = q_{\propto}\sqrt{\frac{k(k+1)}{6N}} = 1.1477$$

Ecuación 34. *Diferencia crítica para el test de Nemenyi.*

Donde k corresponde al número de algoritmos, N corresponde al número de conjuntos de datos y $q_{\propto}$ corresponde a los valores críticos para el test de Nemenyi con 10 clasificadores en $q_{0.05}$ encontrados en [47] y es igual a 3.164. Al calcular las diferencias entre el ranking de un algoritmo con respecto a los otros obtenemos lo que se muestra en la ***Tabla 35*** en donde se encuentra en negrita los valores mayores a la distancia crítica.

Por otro lado, en los conjuntos de datos de regresión se obtuvo el ranking presentado en la ***Tabla 36*** con un valor p de 0.9221 al aplicar las pruebas de Friedman. Este valor de p muestra que el ranking es solo informativo ya que no tiene significancia estadística.

Tabla 35. *Resultados prueba de Nemenyi en problemas de clasificación.*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
IHDELS TT (1)	-	0,2500	**1,7895**	**1,9869**	**2,0132**	**2,4079**	**2,4474**	**2,6421**	**2,7763**	**3,1579**
DECC-G TT (2)		-	**1,5395**	**1,7369**	**1,7632**	**2,1579**	**2,1974**	**2,3921**	**2,5263**	**2,9079**
MOS TT (3)			-	0,1974	0,2237	0,6184	0,6579	0,8526	0,9868	**1,3684**
IHDELS CV (4)				-	0,0263	0,4210	0,4605	0,6552	0,7894	**1,1710**
M-ELM CV (5)					-	0,3947	0,4342	0,6289	0,7631	1,1447
RW TT (6)						-	0,0395	0,2342	0,3684	0,7500
DECC-G CV (7)							-	0,1947	0,3289	0,7105
RW CV (8)								-	0,1342	0,5158
M-ELM TT (9)									-	0,3816
MOS CV (10)										-

Tabla 36. *Prueba de Friedman en problemas de regresión*

Algoritmo	Ranking
MOS TT	**4.4375**
IHDELS TT	4.6875
M-ELM CV	5
MOS CV	5.1875
Random TT	5.1875
DECC-G CV	5.6875
IHDELS CV	5.9375
M-ELM TT	6.125
Random CV	6.3125
DECC-G TT	6.4375

4.2.3 Análisis de los conjuntos de datos

Con el objetivo de observar si las características de los conjuntos de datos de clasificación presentaban alguna variación en los resultados, se realizó un análisis de los resultados según: cantidad de atributos, cantidad de instancias y si estos están o no balanceados. En la ***Tabla 37*** se especifican las características mencionadas anteriormente según cada conjunto de datos, estos se encuentran ordenados alfabéticamente, más adelante se especifica las medidas que se utilizaron para el análisis según la característica.

Tabla 37 *Características de los conjuntos de datos usados para el análisis*

Nombre	Atributos	Instancias	Desbalanceado
Banknote	4	1372	NO
Blood	4	748	SI
Car	6	1728	SI

Cardiotocography	21	2126	SI
Chart	60	600	NO
ClimateSimulation	18	540	SI
Connectionist	60	208	NO
Contraceptive	9	1473	NO
Dermatology	34	366	SI
Diabetes	8	768	NO
Ecoli	7	336	SI
Fertility	9	100	SI
Glass	10	214	SI
Haberman	3	306	SI
Hayes	5	132	NO
Hill	100	606	NO
Indian	10	583	SI
Ionosphere	34	351	NO
Iris	4	150	NO
Knowledge	5	403	NO
Leaf	14	340	NO
Letter	16	20000	NO
Libras	90	360	NO
Optdigits	64	5620	NO
Pen	16	10992	NO
Planning	12	182	SI
QSARBiodegradation	41	1055	NO
Seeds	7	210	NO
Shuttle	9	58000	SI
Spectf	44	269	SI
Vertebral(2C)	6	310	SI
Vertebral(3C)	6	210	SI
Wdbc	30	569	NO
Wilt	5	4839	SI
Wine	13	178	NO
Wine(Red)	11	1599	SI
Yeast	8	1484	SI
Zoo	16	101	NO

4.2.3.1 Distribución según el balance de las clases en los conjuntos de datos

El análisis se realiza a partir de dos grupos formados como siguen:

- Grupo 1: Los conjuntos de datos no están balanceados.
- Grupo 2: Los conjuntos de datos están balanceados.

En la ***Tabla 38*** y la ***Tabla 39*** se presentan los resultados de la prueba de Friedman sobre cada grupo de los conjuntos de datos según su balance, en donde se obtuvo un valor de p de 1.1463E-7 y 0.3436 respectivamente.

Tabla 38 *Prueba de Friedman en grupo 1 según el balance de los conjuntos de datos.*

Algoritmo	Ranking
IHDELS TT	3
DECC-G TT	3.125
IHDELS CV	4.8
MOS TT	5.125
DECC-G CV	5.65
M-ELM CV	6.2
RW TT	6.325
RW CV	6.475
M-ELM TT	6.775
MOS CV	7.525

Tabla 39 *Prueba de Friedman en grupo 2 según el balance de los conjuntos de datos*

Algoritmo	Ranking
IHDELS TT	4.25
DECC-G TT	4.4167
M-ELM CV	4.8333
MOS TT	5.4167
RW TT	5.5278
RW CV	6.0278
M-ELM TT	6.0278
IHDELS CV	6.0556
MOS CV	6.0833
DECC-G CV	6.3611

Como los resultados del primer grupo obtuvieron un valor de p válido, se realizó la prueba post hoc de Nemenyi a estos datos, el valor utilizado como distancia crítica

fue $CD = 3.193$ y los resultados de la prueba se observan en la ***Tabla 40*** y en negrita se muestra donde la diferencia entre rankings es mayor a la distancia crítica.

Tabla 40. *Resultados prueba de Nemenyi grupo 1 (desbalanceados).*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
IHDELS TT (1)	-	0,1250	1,8000	2,1250	2,6500	**3,2000**	**3,3250**	**3,4750**	**3,7750**	**4,5250**
DECC-G TT (2)		-	1,6750	2,0000	2,5250	3,0750	**3,2000**	**3,3500**	**3,6500**	**4,4000**
IHDELS CV (3)			-	0,3250	0,8500	1,4000	1,5250	1,6750	1,9750	2,7250
MOS TT (4)				-	0,5250	1,0750	1,2000	1,3500	1,6500	2,4000
DECC-G CV (5)					-	0,5500	0,6750	0,8250	1,1250	1,8750
M-ELM CV (6)						-	0,1250	0,2750	0,5750	1,3250
RW TT (7)							-	0,1500	0,4500	1,2000
RW CV (8)								-	0,3000	1,0500
M-ELM TT (9)									-	0,7500
MOS CV (10)										-

La ***Tabla 41*** y ***Tabla 42*** muestran los resultados de la prueba de Wilcoxon sobre el grupo 1 y 2 respectivamente. Tener en cuenta que el punto negro indica que el algoritmo de la fila domina al de la columna y el punto blanco que el algoritmo de la columna domina al de la fila.

Tabla 41 *Prueba de Wilcoxon en grupo 1 según el balance de los conjuntos de datos.*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
DECC-G TT (1)	-	•	•		•	•	•	•	•	•
M-ELM TT (2)	○	-		○	○					
RW TT (3)	○		-	○						
IHDELS TT (4)		•	•	-	•	•	•	•	•	•
MOS TT (5)	○	•		○	-		•			•
DECC-G CV (6)	○			○		-				•
M-ELM CV (7)	○			○			-		○	•
RW CV (8)	○			○				-		•
IHDELS CV (9)	○			○					-	•
MOS CV (10)	○			○	○	○			○	-

Tabla 42 *Prueba de Wilcoxon en grupo 2 según el balance de los conjuntos de datos.*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
DECC-G TT (1)	-	•	•							
M-ELM TT (2)		-								
RW TT (3)			-	○						
IHDELS TT (4)				-						
MOS TT (5)					-					
DECC-G CV (6)						-				
M-ELM CV (7)							-		•	
RW CV (8)								-		
IHDELS CV (9)									-	
MOS CV (10)										-

De acuerdo a los resultados obtenidos según el balance de los conjuntos de datos, se puede observar que en los conjuntos de datos no balanceados sus mejores resultados son producto de IHDELS TT y DECC-G TT según la prueba de Friedman, con el post hoc de Nemenyi se establece que no hay una diferencia significativa entre IHDELS TT y DECC-G TT, reafirmándose con la prueba de Wilcoxon dado que entre ellos dos no se establece una dominancia pero sí contra todos los demás, por lo tanto cualquiera de estos dos algoritmos con el modelo entrenamiento-pruebas en conjuntos de datos no balanceados obtienen buenos resultado contra los demás algoritmos comparados.

Por otro lado, en los conjuntos de datos balanceados correspondientes al grupo 2, no se pudo establecer una diferencia significativa de los resultados. Pero de la prueba de Wilcoxon se presenta a DECC-G TT como el algoritmo de mayor dominancia.

4.2.3.2 Distribución según el número de instancias

Para el análisis sobre el número de instancias se consideró lo siguiente:

- Grupo 1: Los conjuntos de datos tienen menos de 450 instancias.
- Grupo 2: Los conjuntos de datos tiene 450 o más instancias.

Dividiendo así entre los conjuntos de datos más pequeños y los más grandes en términos de su número de instancias.

Al realizar la prueba de Friedman se obtuvieron los resultados de ***Tabla 43*** y la ***Tabla 44***, para cada grupo, obteniendo un valor de p de 0.0308 y 3.9477E-7 respectivamente.

Tabla 43 *Prueba de Friedman en grupo 1 según el número de instancias de los conjuntos de datos.*

Algoritmo	Ranking
DECC-G TT	3.8947
IHDELS TT	4.5
DECC-G CV	4.7895
MOS TT	5.0789
IHDELS CV	5.5789
RW CV	5.6053
MOS CV	5.7632
M-ELM CV	5.8421
RW TT	6.7895
M-ELM TT	7.1579

Tabla 44 *Prueba de Friedman en grupo 2 según el número de instancias de los conjuntos de datos.*

Algoritmo	Ranking
IHDELS TT	2.6842
DECC-G TT	3.5789
RW TT	5.1053
IHDELS CV	5.2105
M-ELM CV	5.2632
MOS TT	5.4474
M-ELM TT	5.6842
RW CV	6.9211
DECC-G CV	7.1842
MOS CV	7.9211

Como en ambas pruebas se obtuvo un valor de p válido, se realizó la prueba post hoc de Nemenyi con un valor de distancia crítica $CD = 3.1079$, se utilizó este mismo valor para ambos grupos ya que presentan el mismo valor de N, los resultados de la prueba para el grupo 1 se encuentran en la ***Tabla 45*** y los del grupo 2 en la ***Tabla 46*** y en negrita se muestra las diferencias entre los rankings que son mayores a la distancia crítica.

Tabla 45. *Resultados prueba de Nemenyi grupo 1 (menores a 450 instancias).*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
DECC-G TT (1)	-	0,6053	0,8948	1,1842	1,6842	1,7106	1,8685	1,9474	2,8948	**3,2632**
IHDELS TT (2)		-	0,2895	0,5789	1,0789	1,1053	1,2632	1,3421	2,2895	2,6579
DECC-G CV (3)			-	0,2894	0,7894	0,8158	0,9737	1,0526	2,0000	2,3684
MOS TT (4)				-	0,5000	0,5264	0,6843	0,7632	1,7106	2,0790
IHDELS CV (5)					-	0,0264	0,1843	0,2632	1,2106	1,5790
RW CV (6)						-	0,1579	0,2368	1,1842	1,5526
MOS CV (7)							-	0,0789	1,0263	1,3947
M-ELM CV (8)								-	0,9474	1,3158
RW TT (9)									-	0,3684
M-ELM TT (10)										-

Tabla 46. *Resultados prueba de Nemenyi grupo 2 (mayores a 450 instancias).*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
IHDELS TT (1)	-	0,8947	2,4211	2,5263	2,5790	2,7632	3,0000	**4,2369**	**4,5000**	**5,2369**
DECC-G TT (2)			1,5264	1,6316	1,6843	1,8685	2,1053	**3,3422**	**3,6053**	**4,3422**
RW TT (3)				0,1052	0,1579	0,3421	0,5789	1,8158	2,0789	2,8158
IHDELS CV (4)					0,0527	0,2369	0,4737	1,7106	1,9737	2,7106
M-ELM CV (5)						0,1842	0,4210	1,6579	1,9210	2,6579
MOS TT (6)							0,2368	1,4737	1,7368	2,4737
M-ELM TT (7)								1,2369	1,5000	2,2369
RW CV (8)									0,2631	1,0000
DECC-G CV (9)										0,7369
MOS CV (10)										-

Por otro lado, los resultados de la prueba de Wilcoxon se presentan en la ***Tabla 47*** para el grupo 1 y la ***Tabla 48*** para el grupo 2.

Tabla 47 *Prueba de Wilcoxon en grupo 1 según el número de instancias de los conjuntos de datos.*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
DECC-G CV (1)	-		•				•	•		
M-ELM CV (2)		-								
RW CV (3)			-							
IHDELS CV (4)				-						
MOS CV (5)					-					
DECC-G TT (6)						-	•	•		•
M-ELM TT (7)						○	-		○	○
RW TT (8)						○		-	○	○
IHDELS TT (9)							•	•	-	
MOS TT (10)							•	•		-

Tabla 48 *Prueba de Wilcoxon en grupo 2 según el número de instancias de los conjuntos de datos.*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
DECC-G TT (1)	-	•	•			•		•		•
M-ELM TT (2)	○	-		○		•		•		•
RW TT (3)			-	○		•		•		•
IHDELS TT (4)		•	•	-	•	•	•	•	•	•
MOS TT (5)				○	-	•		•		•
DECC-G CV (6)	○	○		○	○	-	○		○	
M-ELM CV (7)				○		•	-	•		•
RW CV (8)	○			○			○	-	○	
IHDELS CV (9)				○		•		•	-	•
MOS CV (10)	○	○		○	○		○		○	-

En el grupo 1 que corresponde a los conjuntos de datos con menor número de instancias, la prueba de Friedman con el post-hoc de Nemenyi muestra que no existen diferencias significativas en los resultados, es decir ni entre los algoritmos

ni entre los modelos de validación. Sin embargo, la prueba de Wilcoxon arroja a DECC-G TT como el algoritmo de mayor dominancia contra tres algoritmos.

Los conjuntos de datos con mayor número de instancias, es decir el grupo 2, con la prueba de Friedman arroja que tres algoritmos implementados con entrenamiento-pruebas ocupan los primeros lugares y tres algoritmos implementados con validación cruzada ocupan los tres últimos; del análisis post hoc de Nemenyi se obtiene que IHDELS TT y DECC-G TT tienen una diferencia significativa con respecto a los 3 últimos algoritmos del ranking implementados con validación cruzada. Adicionalmente la prueba de Wilcoxon ubica a IHDELS TT como el algoritmo de mayor dominancia.

4.2.3.3 Distribución según el número de atributos

El análisis sobre el número de atributos de cada conjunto de datos se hizo distribuyéndolos en dos grupos, así:

- Grupo 1: Los conjuntos de datos tienen menos de 20 atributos.
- Grupo 2: Los conjuntos de datos tiene 20 o más atributos.

La ***Tabla 49*** y ***Tabla 50*** presentan los resultados obtenidos con la prueba de Friedman en cada grupo. Para el grupo 1 se obtuvo un valor de p 3.1856E-4 y para el grupo 2 el valor de p fue de 0.0655.

Tabla 49 *Prueba de Friedman en grupo 1 según el número de atributos de los conjuntos de datos.*

Algoritmo	Ranking
IHDELS TT	3.3704
DECC-G TT	3.9519
MOS TT	5.3519
M-ELM CV	5.4815
IHDELS CV	5.6481
RW CV	5.963
RW TT	6.1111
DECC-G CV	6.1852
M-ELM TT	6.3889
MOS CV	6.6481

Tabla 50 *Prueba de Friedman en grupo 2 según el número de atributos de los conjuntos de datos.*

Algoritmo	Ranking
DECC-G TT	3.4545
IHDELS TT	4.1364
IHDELS CV	4.7727
MOS TT	5.0455
DECC-G CV	5.5
RW TT	5.5455

M-ELM CV	5.7273
M-ELM TT	6.5
RW CV	7
MOS CV	7.3182

Como los resultados del primer grupo obtuvieron un valor de p válido, se realizó l prueba post hoc de Nemenyi con un valor de distancia crítica $CD = 2.6072$, los resultados de la prueba se muestran en la ***Tabla 51*** y en negrita se muestra la diferencia entre rankings que tiene un valor mayor a la distancia crítica.

Tabla 51. *Resultados prueba de Nemenyi grupo 1 (menos de 20 atributos).*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
IHDELS TT (1)	-	0,5815	1,9815	2,1111	2,2777	2,5926	**2,7407**	**2,8148**	**3,0185**	**3,2777**
DECC-G TT (2)		-	1,4000	1,5296	1,6962	2,0111	2,1592	2,2333	2,4370	**2,6962**
MOS TT (3)			-	0,1296	0,2962	0,6111	0,7592	0,8333	1,0370	1,2962
M-ELM CV (4)				-	0,1666	0,4815	0,6296	0,7037	0,9074	1,1666
IHDELS CV (5)					-	0,3149	0,4630	0,5371	0,7408	1,0000
RW CV (6)						-	0,1481	0,2222	0,4259	0,6851
RW TT (7)							-	0,0741	0,2778	0,5370
DECC-G CV (8)								-	0,2037	0,4629
M-ELM TT (9)									-	0,2592
MOS CV (10)										-

En la ***Tabla 52*** y la ***Tabla 53*** se evidencia el resultado de la prueba de Wilcoxon sobre cada grupo según el número de atributos en los conjuntos de datos.

Tabla 52 *Prueba de Wilcoxon en grupo 1 según el número de atributos de los conjuntos de datos.*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
DECC-G CV (1)	**-**					○			○	
M-ELM CV (2)		**-**							○	
RW CV (3)			**-**			○			○	
IHDELS CV (4)				**-**					○	
MOS CV (5)					**-**	○			○	
DECC-G TT (6)	•					**-**	•	•		•
M-ELM TT (7)						○	**-**		○	○
RW TT (8)						○		**-**	○	○
IHDELS TT (9)	•		•	•			•	•	**-**	•
MOS TT (10)							•	•	○	**-**

Tabla 53 *Prueba de Wilcoxon en grupo 2 según el número de atributos de los conjuntos de datos.*

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
DECC-G CV (1)	-									
M-ELM CV (2)		-								
RW CV (3)			-							
IHDELS CV (4)				-		○				
MOS CV (5)					-	○				○
DECC-G TT (6)						-	•			
M-ELM TT (7)						○	-		○	
RW TT (8)								-		
IHDELS TT (9)									-	
MOS TT (10)										-

El grupo 1 que representa los conjuntos de datos con menor número de atributos obtiene en la prueba de Friedman un valor de p válido que ubica a IHDELS TT y DECC-G TT como los mejores algoritmos, sin embargo, en la prueba post hoc de Nemenyi se observa que no existe una diferencia significativa entre estos dos. Por otro lado, la prueba de Wilcoxon ubica a IHDELS TT como el algoritmo con mayor dominancia.

Del grupo 2 no se pudo establecer un ranking significativo dado que no hay un valor de P válido; de la prueba de Wilcoxon DECC-G TT se presenta como el algoritmo de mayor dominancia.

4.2.4 Análisis de resultados

A partir de los datos, se pudo observar que los algoritmos meta-heurísticos: IHDELS y DECC-G implementados con entrenamiento-pruebas presentan un mejor desempeño general en la optimización de los pesos de entrada y sesgos de una SLFN entrenada con ELM frente al algoritmo del estado del arte M-ELM y la caminata aleatoria.

En los problemas de clasificación el algoritmo IHDELS implementado con el modelo de entrenamiento-pruebas presentó mejores resultados, teniendo en cuenta que en la prueba de Friedman es el primero en el ranking, sin embargo, en la prueba de Nemenyi se observa que no tiene una diferencia significativa con el algoritmo DECC-G TT, pero al realizar el análisis con la prueba de Wilcoxon IHDELS TT es quien presenta mayor dominancia.

Los problemas de regresión no pudieron establecer un nivel de dominancia entre ellos ni un valor P en la prueba de Friedman que permita determinar que los valores obtenidos sean estadísticamente significativos; no obstante, MOS se presenta como el mejor algoritmo con el modelo de validación de entrenamiento-prueba, mientras que en validación cruzada los mejores resultados pertenecen a M-ELM.

Los mejores resultados en problemas de clasificación siempre fueron consecuencia de utilizar el modelo de entrenamiento-prueba. Se considera que esto se debe principalmente al número de soluciones diferentes que se generan usando TT, ya que realmente se crean 3.000 al igual que las EFOs establecidas para los algoritmos, mientras que para CV con k=10 sólo se crean 300, ya que cada evaluación por CV realiza 10 EFOs.

Los problemas de clasificación demuestran más confiabilidad en la información que proveen puesto que se pudo establecer un nivel de dominancia entre ellos y un ranking de Friedman con diferencias significativas utilizando el post hoc de Nemenyi que asegura que los datos son estadísticamente válidos y que ubica a IHDELS TT y a DECC-G TT como los mejores algoritmos entre los que fueron evaluados; a diferencia de los problemas de regresión de los cuales no se pudo establecer un nivel de dominancia ni un ranking válido.

Capítulo 5

5. CONCLUSIONES Y TRABAJO FUTURO

5.1 CONCLUSIONES

El presente trabajo de investigación adaptó los tres algoritmos que presentaron mejores resultados en la competencia 2015 IEEE CEC en problemas de alta dimensionalidad, a saber: coevolución cooperativa basada en evolución diferencial (DECC-G), muestreo múltiple de descendientes (MOS) e hibridación iterativa de evolución diferencial con búsqueda local y reinicios (IHDELS). Estos algoritmos fueron utilizados para optimizar la asignación de pesos de entrada y sesgos de una SLFN entrenada con ELM y fueron probados sobre problemas de clasificación y regresión familiares para la comunidad científica. La experimentación se llevó a cabo con un número de evaluaciones de la función objetivo que mantuviera el principal concepto de una ELM el cual es realizar el entrenamiento de la red neuronal en una cantidad de tiempo menor comparado con algoritmos como el de propagación hacía atrás. El presente trabajo pudo determinar que los algoritmos meta-heurísticos sí presentan mejores resultados, donde los mejores se obtuvieron con IHDELS, además se determinó que el modelo de entrenamiento-prueba supera al de validación cruzada en el problema de la ELM, lo cual se confirmó a través de las pruebas no paramétricas de Friedman y Wilcoxon.

A partir de los datos extraídos de la experimentación se demostró que el algoritmo del estado del arte M-ELM fue superado en problemas de clasificación utilizando el modelo de entrenamiento-prueba por los dos algoritmos meta-heurísticos implementados en el siguiente orden: 1) IHDELS y 2) DECC-G. Por otro lado, en los problemas de regresión no se pudo establecer con certeza cuál es el algoritmo ganador dado que en la prueba de Friedman no se obtuvo un valor p menor a 0.05 y la prueba de Wilcoxon no pudo establecer la dominancia de ninguno de los algoritmos, esto sugiere que no existe una cantidad significativa de datos para determinar cuál es la mejor meta-heurística a utilizar, lo cual por motivos de alcance del proyecto no pudieron ser llevados a cabo. Sin embargo, con la información que se cuenta en los problemas de regresión la prueba de Friedman indica que el algoritmo que está presentando hasta ahora los mejores resultados es MOS utilizando el modelo de entrenamiento-prueba.

El éxito de IHDELS radica en que utiliza una estrategia de exploración a través del algoritmo evolutivo y otra de explotación por medio de las búsquedas locales, por lo que se comporta como un algoritmo memético (aunque no incluye conocimiento específico del problema). Su mecanismo de reinicio establece una tasa de mejora que le permite realizar nuevamente una exploración sobre el espacio de búsqueda evitando que caiga en un óptimo local. También, las estrategias de optimización que implementa cada búsqueda local tienen un enfoque diferente, lo que ayuda a

recorrer el espacio de búsqueda por diferentes caminos en las dimensiones del problema. El algoritmo IHDELS utiliza originalmente en [9] como búsquedas locales al algoritmo MTS-LS1 y L-BFGS-B, no obstante el cálculo de la derivada que era necesario para implementar el algoritmo L-BFGS-B consumía rápidamente el número máximo de evaluaciones de la función objetivo en el problema específico del proyecto, con esto en mente se optó por sustituirlo con el algoritmo de ascenso a la colina (hill climbing) que logró ser una buena opción.

Los resultados obtenidos en este trabajo demuestran que los algoritmos meta-heurísticos, específicamente IHDELS y DECC-G sí presentan mejores resultados en la exactitud del entrenamiento de la SLFN con ELM que el algoritmo del estado del arte M-ELM. Este trabajo además es el primero que se realiza para optimizar los pesos de entrada y sesgos de una ELM usando algoritmos de optimización de alta dimensionalidad. Las comparaciones en los tiempos de ejecución de los algoritmos no pudieron establecerse, dado que las características de los computadores a nivel de software no eran homogéneas, disminuyendo el rendimiento en algunas máquinas, esto se observó durante la experimentación donde al asignar la misma tarea a dos computadores estos presentaban una diferencia considerable en tiempo.

El uso de algoritmos especializados en alta dimensionalidad utilizados tiene un alto grado de dificultad en el afinamiento de sus parámetros, dado que estos deben ser ajustados conjuntamente con los parámetros de sus algoritmos subordinados generando un mayor conjunto de combinaciones entre estos. Además, en algunos casos es necesario definir manualmente los parámetros para garantizar que todas las características de la meta-heurística son aprovechadas al completar al menos una iteración; el cual fue el caso de DECC-G donde la definición de la cantidad de subcomponentes, las evaluaciones en SaNSDE y en DE debían ser meticulosamente seleccionadas.

Entre los resultados validos obtenidos en la división de los conjuntos de datos por: número de instancias, número de atributos y el balance de estos, se observó que IHDELS y DECC-G implementados con el modelo de validación de entrenamiento-pruebas siempre lograban el mejor desempeño, sin embargo para realizar un análisis más profundo sobre el comportamiento de los algoritmos según las características de los conjuntos de datos se recomienda utilizar técnicas de clustering que agrupen y describan apropiadamente los subconjuntos de estos.

5.2 TRABAJO FUTURO

En este trabajo de investigación se utilizaron tres meta-heurísticas, las que mostraron los mejores resultados en la competencia 2015 IEEE CEC, por lo que se espera realizar los mismos experimentos con los algoritmos ganadores en funciones de alta dimensionalidad de la siguiente competencia IEEE CEC, además se espera utilizar algunos algoritmos no especializados en alta dimensionalidad, pero efectivos en un número pequeño de evaluaciones de la función objetivo.

También se hace interesante optimizar la cantidad de neuronas de la capa oculta junto a los valores de los pesos de entrada y sesgos de la ELM, con los algoritmos propuestos, para así no tener una cantidad de neuronas en la capa oculta que sean innecesarias según el problema que se está evaluando, evitando el sobre entrenamiento de algunos conjuntos de datos.

Sería conveniente realizar un mayor trabajo en la selección de las técnicas y búsquedas locales para las meta-heurísticas MOS e IHDELS respectivamente, donde estas no se limiten a tener únicamente dos técnicas y dos búsquedas locales, si no que se puedan integrar más técnicas o búsquedas locales permitiendo que las optimizaciones se hagan con diferentes enfoques.

Además, como los conjuntos de datos presentan diferente distribución en la cantidad de clases se podría realizar un trabajo donde se haga el análisis sobre subconjuntos de estos con el objetivo de observar el comportamiento de los algoritmos en conjuntos de datos con características similares.

Finalmente sería apropiado hacer un análisis exhaustivo sobre los resultados según las características de los conjuntos de datos para observar como se comportan los modelos de validación y los algoritmos según las diversas agrupaciones.

Capítulo 6

6. BIBLIOGRAFIA

[1] E. Cambria *et al.*, “Extreme Learning Machines,” *IEEE Intell. Syst.*, vol. 28, no. 6, pp. 30–59, 2013.

[2] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme Learning machine: a new learning scheme of feedforward neural networks,” *Proc. IEEE Int. Jt. Conf. Neural Networks*, vol. 2, pp. 985–990, 2004.

[3] P. A. Henríquez and G. A. Ruz, “Extreme learning machine with a deterministic assignment of hidden weights in two parallel layers,” *Neurocomputing*, vol. 226, no. June 2016, pp. 109–116, 2017.

[4] Y. Zhang, J. Wu, Z. Cai, P. Zhang, and L. Chen, “Memetic Extreme Learning Machine,” *Pattern Recognit.*, vol. 58, pp. 135–148, 2016.

[5] G. Huang, G. Bin Huang, S. Song, and K. You, “Trends in extreme learning machines: A review,” *Neural Networks*, vol. 61, pp. 32–48, 2015.

[6] T. Matias, F. Souza, R. Araújo, and C. H. Antunes, “Learning of a single-hidden layer feedforward neural network using an optimized extreme learning machine,” *Neurocomputing*, vol. 129, pp. 428–436, 2014.

[7] A. LaTorre, S. Muelas, and J. M. Peña, “Multiple offspring sampling in large scale global optimization,” *WCCI 2012 IEEE World Congr. Comput. Intell.*, 2012.

[8] Z. Yang, K. Tang, and X. Yao, “Large scale evolutionary optimization using cooperative coevolution,” *Inf. Sci. (Ny).*, vol. 178, no. 15, pp. 2985–2999, 2008.

[9] D. Molina and F. Herrera, “Hibridación iterativa de DE con búsqueda local con reinicio para problemas de alta dimensionalidad.”

[10] C. Cobos, C. Daza, C. Martinez, and M. Mendoza, “Calibration of microscopic trafic flow simulation models using a memetic algorithm with solis and wets local search chaining (MA-SW-Chains),” *Adv. Artif. Intell. IBERAMIA 2016*, pp. 365–375, 2016.

[11] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, 1997.

[12] D. J. Montana and L. Davis, “Training feedforward neural networks using genetic algorithms,” *11th Int. Jt. Conf. Artif. Intell.*, vol. 89, pp. 762–767, 1989.

[13] P. Sharma et al., “Feedforward Neural Network : A Review,” *Int. J. Adv. Res. Eng. Appl. Sci.*, vol. 2, no. 10, pp. 25–34, 2013.

[14] Y. Chauvin and D. E. Rumelhart, *Backpropagation: theory, architectures, and applications*. Lawrence ERLBaum Associates, 1995.

[15] G.-G. Wang, M. Lu, Y.-Q. Dong, and X.-J. Zhao, “Self-adaptive Extreme

Learning Machine," *Neural Comput. Appl.*, vol. 27, 2015.

[16] Z. Beheshti and S. M. H. Shamsuddin, "A review of population-based meta-heuristic algorithm," *Int. J. Adv. Soft Comput. its Appl.*, vol. 5, no. 1, 2013.

[17] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms: Second Edition*. Cambrdige, 2010.

[18] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms: Second Edition*. Cambrdige, 2010.

[19] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling Up Fast Evolutionary Programming with Cooperative Coevolution," *2001 Congr. Evol. Comput.*, vol. 2, pp. 1101–1108, 2001.

[20] Z. Yang, K. Tang, and X. Yao, "Self-adaptive Differential Evolution with Neighborhood Search," *2008 IEEE Congr. Evol. Comput.*, pp. 1110–1116, 2008.

[21] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *J. Glob. Optim.*, vol. 11, no. 4, pp. 341–359, 1997.

[22] F. J. Solis and R. J.-B. Wets, "Minimization By Random Search Techniques," *Math. Oper. Res.*, vol. 6, no. 1, pp. 19–30, 1981.

[23] L. Y. Tseng and C. Chen, "Multiple trajectory search for large scale global optimization," *2008 IEEE Congr. Evol. Comput.*, pp. 3052–3059, 2008.

[24] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A Limited Memory Algorithm for Bound Constrained Optimization," *SIAM J. Sci. Comput.*, vol. 16, no. 5, pp. 1190–1208, 1995.

[25] Y. Fei, G. Rong, B. Wang, and W. Wang, "Parallel L-BFGS-B algorithm on GPU," *Comput. Graph.*, vol. 40, no. 1, pp. 1–9, 2014.

[26] S. Luke, *Essentials of Metaheuristics*. 2013.

[27] A. J. Nebro and J. J. Durillo, "JMetal: A Java framework for multi-objective optimization," *Adv. Eng. Softw.*, vol. 42, pp. 760–771, 2011.

[28] Q.-Y. Zhu, A. K. Qin, P. N. Suganthan, and G.-B. Huang, "Evolutionary extreme learning machine," *Pattern Recognit.*, vol. 38, no. 10, pp. 1759–1763, 2005.

[29] R. Ghosh and B. Verma, "A Hierarchical Method For Finding Optimal Architecture And Weight Using Evolutionary Least Square Based Learning," *Int. J. Neural Syst.*, vol. 13, no. 1, pp. 13–24, 2003.

[30] Y. Xu and Y. Shu, "Evolutionary Extreme Learning Machine – Based on Particle Swarm Optimization," *Adv. Neural Networks*, no. 2, pp. 644-- 652, 2006.

[31] N. Liu and H. Wang, "Ensemble based extreme learning machine," *IEEE Signal Process. Lett.*, vol. 17, no. 8, pp. 754–757, 2010.

[32] S. Saraswathi, S. Sundaram, N. Sundararajan, M. Zimmermann, and M. Nilsen-Hamilton, “ICGA-PSO-ELM approach for accurate multiclass cancer classification resulting in reduced gene sets in which genes encoding secreted proteins are highly represented,” *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, vol. 8, no. 2, pp. 452–463, 2011.

[33] D. N. G. Silva, L. D. S. Pacifico, and T. B. Ludermir, “An evolutionary extreme learning machine based on group search optimization,” pp. 574–580, 2011.

[34] I. Valença and M. Valença, “Optimizing the Extreme Learning Machine Using Harmony Search for Hydrologic Time Series Forecasting,” *Intell. Data Eng. Autom. Learn. - IDEAL 2012*, pp. 261–269, 2012.

[35] K. LI, R. WANG, S. KWONG, and J. CAO, “Evolving Extreme Learning Machine Paradigm With Adaptive Operator Selection and Parameter Control,” *Int. J. Uncertainty, Fuzziness Knowledge-Based Syst.*, vol. 21, no. supp02, pp. 143–154, 2013.

[36] L. Pacifico and T. B. Ludermir, “Evolutionary Extreme Learning Machine Based on Particle Swarm Optimization and Clustering Strategies,” vol. 560, pp. 305–310, 2013.

[37] Y. Bazi, N. Alajlan, F. Melgani, H. AlHichri, S. Malek, and R. R. Yager, “Differential Evolution Extreme Learning Machine for the Classification of Hyperspectral Images,” *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 6, pp. 1066–1070, 2014.

[38] R. Dash, P. K. Dash, and R. Bisoi, “A self adaptive differential harmony search based optimized extreme learning machine for financial time series prediction,” *Swarm Evol. Comput.*, vol. 19, pp. 25–42, 2014.

[39] D. E. Pusil Rojas, “Algoritmo Meta-Heurístico Mono Objetivo para Optimizar el Proceso de Entrenamiento de una Máquina de Aprendizaje Extremo basado en jMetal y Variaciones de la Búsqueda Armónica,” Universidad del Cauca, 2018.

[40] H. Kong, “Evolving extreme learning machine paradigm with adaptive operator selection and parameter control,” *Int. J. Uncertainty, Fuzziness Knowledge-Base Syst.*, vol. 21, no. December, pp. 143–154, 2013.

[41] P. Rafaelizadeh, L. Tang, and H. Liu, “Cross-Validation,” *Encycl. Database Syst.*, pp. 532–538, 2009.

[42] J. Schneider, “Cross Validation,” 1997. [Online]. Available: http://www.cs.cmu.edu/~schneide/tut5/node42.html.

[43] K. V Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*, vol. 28. 2005.

[44] S. Luke, *Essentials of Metaheuristics*. 2015.

[45] A. K. Qin and P. N. Suganthan, “Self-adaptive Differential Evolution Algorithm for Numerical Optimization,” *2005 IEEE Congr. Evol. Comput.*, pp.

1785–1791, 2005.

[46] J. Torres Jimenez and E. Rodriguez Tello, “New bounds for binary covering arrays using simulated annealing,” *Inf. Sci. (Ny).*, vol. 185, no. 1, pp. 137–152, 2012.

[47] J. Demsar, “Statitistical Comparison of Classifiers over Multiple Data Sets,” *Mach. Learn. Res.*, pp. 1–30, 2006.