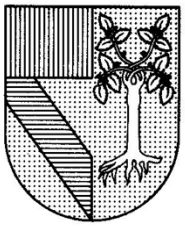

UNIVERISDAD PANAMERICANA
FACULTAD DE INGENIERÍA
Con estudios incorporados a la
Secretaría de Educación Pública

“Reconocimiento de Actividades Humanas con grandes datos: Algoritmo de Festín de Pirañas para escalabilidad y entrenamiento de Redes de Hidrocarburos Artificiales”

T E S I S

QUE PARA OBTENER EL TÍTULO DE

MAESTRÍA EN INGENIERÍA

P R E S E N T A

JOSÉ GUILLERMO GONZALEZ MORA

DIRECTOR DE TESIS:

DR. HIRAM EREDÍN PONCE ESPINOSA

CDMX 2020

UNIVERSIDAD PANAMERICANA

Resumen

Facultad de Ingeniería

Universidad Panamericana

Maestro en Ingeniería

Reconocimiento de Actividades Humanas con grandes datos: Algoritmo de Festín de Pirañas para escalabilidad y entrenamiento de Redes de Hidrocarburos Artificiales

by Ing. José Guillermo GONZÁLEZ MORA

En la actualidad el Reconocimiento de Actividades Humanas es un problema de investigación abierto con impacto en áreas médicas, de entretenimiento y seguridad entre otras. Algunos de los mayores retos de este problema se encuentran en el manejo de conjuntos de datos con gran número de dimensiones y en la flexibilidad de los modelos de clasificación. Se han explorado soluciones utilizando aprendizaje automático, sobre los que destaca el algoritmo de Redes de Hidrocarburos Artificiales por su robustez en la aproximación de soluciones para fenómenos con ruido presente y alta variación entre dimensiones. Actualmente el modelo utiliza un algoritmo basado en el gradiente para su entrenamiento, volviéndolo preciso en aprendizaje de problemas con poca cantidad de dimensiones y registros para el entrenamiento, pero dificultando su escalabilidad. La solución propuesta en este documento pretende integrar un nuevo algoritmo metaheurístico y una estrategia de procesamiento paralelo para el entrenamiento de este modelo sobre una base de datos de más de 800 dimensiones. El algoritmo presentado fue inspirado en el comportamiento de los cardumenes de pirañas al momento de alimentarse, por lo que recibe el nombre de Algoritmo de Festín de Pirañas. El algoritmo incorpora dos funciones de actualización de movimiento, cuatro hiper-parámetros ajustables entre los cuales existe un control de puntos de interés y una estrategia de incentivos y castigos para la política de exploración del algoritmo. Esta integración logró disminuir el tiempo de entrenamiento en aproximadamente 95 % sin perder precisión, puntuación F1 y exactitud respecto a pruebas anteriores del modelo; presentando 0.94 de exactitud, 0.89 de puntuación F1 y 0.89 de precisión promedio en la clasificación de la base de datos “UP-Fall Detection DataSet”

Índice general

Resumen III

1. Introducción 1

2. Antecedentes y estado de la cuestión 5
2.1. El modelo de Redes de Hidrocarburos Artificiales . . . 5
2.2. Métodos de optimización . . . 6
2.2.1. Algoritmos de optimización metaheurística . . . 8
Algoritmos genéticos - GA . . . 8
Algoritmo de optimización por parvada - PSO . . . 8
Algoritmo de caza del lobo gris – GWO . . . 9
Algoritmo de murciélago - BA / BAT . . . 9
2.2.2. Ventajas y desventajas . . . 9
2.2.3. Selección de un método de optimización . . . 10
2.3. Escalabilidad del aprendizaje automático . . . 11
2.3.1. Procesamiento paralelo y cómputo distribuido . . . 11
2.4. Reconocimiento de actividades humanas - HAR . . . 13
2.4.1. Aprendizaje automático en el Reconocimiento de Actividades Humanas . . . 13

3. Descripción de la propuesta 17
3.1. Algoritmo de festín de pirañas - PFO . . . 19
3.1.1. Inspiración . . . 19
3.1.2. Diseño del algoritmo . . . 20
3.1.3. Creación del cardumen . . . 22
3.1.4. Inicialización y actualización de posiciones de las pirañas . . . 23
3.1.5. Fase de ataque . . . 24
3.1.6. Fase de navegación . . . 25
3.2. Pseudocódigo de la propuesta . . . 26
3.3. Paralelización . . . 26
3.4. Codificación del algoritmo . . . 28
3.5. Métricas y soporte de la propuesta . . . 29
3.6. Diseño de experimentos . . . 32
3.7. Implementación y ambiente de pruebas . . . 33
3.7.1. Hardware: . . . 33
3.7.2. Software: . . . 33

4. Resultados y discusión 35
4.1. Análisis de impacto en el tiempo de ejecución . . . 36

4.1.1. Experimentación serializada 36
4.1.2. Experimentación paralelizada 38
4.2. Análisis del error cuadrático medio 39
4.2.1. Experimentación serializada 41
4.2.2. Experimentación paralelizada 44
4.3. Caso de estudio: aplicación de PFO y AHN en HAR 45
4.4. Discusión y resultados del algoritmo PFO 53

5. Conclusiones y trabajo futuro **57**
5.1. Conclusiones 57
5.2. Trabajo futuro 58

6. Contribuciones **61**

A. Tablas de resultados **63**

Bibliografía **71**

Capítulo 1

Introducción

El modelo de Redes de Hidrocarburos Artificiales, o AHN por sus siglas en inglés, forma parte de los modelos de aprendizaje automático provenientes de Redes Orgánicas Artificiales; es un modelo de aprendizaje automatizado que ha demostrado estabilidad algorítmica y buenos resultados en filtrado de señales y control en sistemas con incertidumbre [42] [41] [40]. En la actualidad el modelo es entrenado por un algoritmo de optimización basado en el gradiente que presenta buenos resultados ante aprendizaje realizado sobre funciones con pocas dimensiones y datos. En la realidad, la mayoría de los fenómenos donde las características de flexibilidad y robustez del modelo pueden ser utilizadas, son fenómenos de gran dimensionalidad y un gran número de registros. Por esta razón, se ha recurrido al uso de algoritmos metaheurísticos para el entrenamiento de los modelos de aprendizaje automático [26] [52] [11]. Dichos algoritmos tienden a encontrar máximos o mínimos globales de una función objetivo. En el caso del aprendizaje automático, esta función objetivo será la función de error resultante de la comparación de los datos reales contra los aproximados por el modelo. Dentro de las aplicaciones en las que ha sido empleado el modelo de AHN se encuentra el reconocimiento de actividades humanas, o HAR por sus siglas en inglés [40]. El modelo de AHN destaca por sus resultados de flexibilidad y robustez a ruido en sensores [40].

El problema principal discutido en este documento es la escalabilidad del modelo de Redes de Hidrocarburos artificiales al enfrentarse a problemas con gran cantidad de dimensiones y de observaciones. Siendo de especial relevancia el impacto computacional de la implementación de un algoritmo metaheurístico. Centrándose en los algoritmos metaheurísticos poblacionales que, al utilizar múltiples agentes que se desplazan por el espacio de la función objetivo con cierta aleatoriedad, incrementan la cantidad de cálculos necesarios para lograr la optimización [37]. Como lo explica Ron Bekkerman en el libro “Scaling up Machine Learning: Parallel and Distributed Approaches” [3], existen 6 características en los problemas de aprendizaje automático, mostradas a continuación, que convierten la aplicación de estos modelos en una tarea muy retadora para sistemas locales y serializados.

1. **Gran cantidad de observaciones**: Referente a la cantidad de observaciones que conforman a la base de datos de entrenamiento o de pruebas.
2. **Gran cantidad de dimensiones de entrada**: La cantidad de variables o dimensiones que forman a cada observación.

3. **Complejidad del modelo y del algoritmo**: Algunos modelos de aprendizaje funcionan a través de algoritmos complejos o con subrutinas computacionalmente costosas.
4. **Tiempo de inferencia**: Aplicaciones que dependen de predicciones en tiempo real como navegación o robótica de precisión requieren de predicciones en linea con baja latencia.
5. **Predicciones secuenciales o interdependientes**: Aplicaciones cuya resolución final depende de múltiples predicciones pasadas.
6. **Selección del modelo**: Existen modelos que por su funcionamiento y subrutinas lógicas no son escalables.

Este trabajo se concentra en tres de las principales problemáticas, que son: cantidad de instancias u observaciones de datos, la cantidad de dimensiones de entrada y la complejidad algorítmica del motor de entrenamiento. La cantidad de observaciones es relevante ya que el modelo de Redes de Hidrocarburos Artificiales necesita de gran cantidad de registros para su correcto entrenamiento pero en su implementación actual tomaría mucho tiempo entrenarlo con grandes cantidades de observaciones. En segundo lugar, la cantidad de dimensiones que forman cada observación es importante según la aplicación. En este caso, el problema de Reconocimiento de Actividades Humanas puede tener cientos de dimensiones dada la cantidad de métricas necesarias para distinguir una categoría de otra. Por último, la complejidad del algoritmo actual radica en ser un método de optimización basado en el gradiente de la función de búsqueda. Encontrar la derivada de una función compuesta por cientos de dimensiones es un proceso costoso en términos computacionales. Esto conduce a inestabilidad de soluciones y a largos tiempos de entrenamiento. Las tres características restantes no son objeto de estudio en la tesis ya que no se busca implementar un sistema de predicciones en línea, la tarea de clasificación abordada no es interdependiente ni secuencial y el modelo ya ha sido seleccionado por sus resultados en aplicaciones anteriores. El objetivo general de este trabajo es el mejoramiento de la escalabilidad del modelo de Redes de Hidrocarburos Artificiales, disminuyendo el tiempo de entrenamiento sin afectar negativamente sus ventajas en flexibilidad, precisión y estabilidad en la tarea de reconocimiento de actividades humanas; que es un problema de clasificación multiclase.

El reconocimiento de actividades humanas, consiste en la identificación de una o varias actividades realizadas por un ser humano [26]. Este reconocimiento puede ser realizado con sensores periféricos o vesitibles y el proceso de identificación es realizado ya sea en línea o fuera de línea. Es importante volver a evaluar el modelo de AHN en la aplicación de HAR ya que las ventajas resultan pertinentes para una aplicación real, pero está limitado por problemas de escalabilidad. Debido a esto, se han explorado distintas estrategias de procesamiento que permitan disminuir el tiempo necesario para encontrar la solución aproximada. Entre dichas técnicas, destaca la estrategia de implementar procesamiento paralelo [3], lo que permite a múltiples unidades de procesamiento ejecutar operaciones al mismo tiempo y posteriormente unificar resultados para acelerar el tiempo de procesamiento.

El primer objetivo específico de este desarrollo está en la implementación de una estrategia de procesamiento paralelo, con lo que podrá reducirse el tiempo de entrenamiento

total del modelo. El segundo objetivo específico es modificar el método de entrenamiento del modelo AHN, integrándole un algoritmo metaheurístico, con el fin de mejorar o mantener los índices de precisión, exactitud y puntuación F1. El tercer objetivo específico está relacionado con el diseño de una nueva propuesta de algoritmo de optimización metaheurística que permita el control de puntos de interés en un espacio de búsqueda, mejorando su adaptabilidad para tareas de entrenamiento. Con estos objetivos establecidos, se propone la siguiente hipótesis. Mediante la integración de un nuevo algoritmo metaheurístico como motor de entrenamiento del modelo de Redes de Hidrocarburos Artificiales y una estrategia de procesamiento paralelo, se logrará disminuir el tiempo de entrenamiento total sin disminuir la precisión, estabilidad y puntuación F1 del modelo. La comparación se realiza respecto a pruebas anteriores del modelo entrenado con un algoritmo basado en el gradiente y con una estrategia de procesamiento serial. La aplicación del modelo modificado será sobre la base de datos "UP Fall-Detection Dataset" [29] ya que presenta características propias de un fenómeno real con múltiples dimensiones y gran cantidad de datos, además de ser un problema vigente en el área médica y social.

La metodología de la propuesta que se sigue en el presente documento es:

1.- Investigación inicial: En esta etapa se realizó la investigación del estado actual del modelo de AHN. Se puso especial atención a resultados de las últimas aplicaciones así como a las características de la implementación actual.

2.- Diagnostico del problema: Durante la investigación inicial se observó que los casos de uso del modelo hasta la fecha, se encontraban enfocados a aplicaciones con conjuntos de datos controlados y se planteó la posibilidad de ser aplicado a conjuntos de datos masivos. Fue recopilada información sobre el modelo original y el algoritmo de entrenamiento basado en el gradiente y se determinó la problemática originada por sus limitantes.

3.- Formulación de objetivo e hipótesis: Se propuso como objetivo lograr escalar el modelo AHN para mejorar la precisión y tiempo de ejecución en la etapa de entrenamiento. Esto se lograría a través de la implementación de una estrategia de procesamiento paralelo y la integración de la optimización metaheurística como motor de entrenamiento.

4.- Investigación del estado del arte: Se realizó trabajo de investigación sobre técnicas de escalamiento de modelos de aprendizaje automático, alternativas e impacto. Por otra parte, se investigaron también otros algoritmos de optimización capaces de escalar a la par de problemas de gran dimensionalidad. Se encontraron dos altenativas de escalamiento y múltiples aplicaciones de algoritmos metaheurísticos de optimización para otros modelos de aprendizaje automático y se seleccionaron los más relevantes.

5.- Implementación: Se evaluaron los recursos tecnológicos disponibles para el desarrollo de la propuesta, así como la documentación y estabilidad de las tecnologías seleccionadas. Se preparó un ambiente de desarrollo y pruebas con los recursos y tecnologías disponibles y posteriormente fue desarrollado el código en Python de AHN, la integración con los algoritmos metaheurísticos seleccionados y el desarrollo para compatibilidad con CUDA

6.- Pruebas de estabilidad y resultados preliminares: Una vez terminado el desarrollo del ambiente de pruebas, se realizaron pruebas de estabilidad, depuración del código y fueron probadas las agendas de ejecución y recopilación de resultados sobre funciones de prueba. Esta experimentación es la métrica base para la comparación de resultados posteriores.

7.- Experimentación: Se recogieron métricas de tiempo de ejecución y error cuadrático medio de los experimentos realizados.

8.- Análisis de resultados: Basado en los resultados de los experimentos realizados, se desarrolló la propuesta de algoritmo metaheurístico diseñado específicamente para el modelo de AHN. La inspiración se dio a partir de la investigación del comportamiento y dinámicas sociales de las pirañas de abdomen rojo. El nombre del algoritmo propuesto es "Algoritmo de Festín de Pirañas" o PFO por las siglas de su traducción al inglés "Piranha Feast Optimization".

9.- Resultados de la propuesta: Una vez implementado el algoritmo se repitió la experimentación anterior y se recopilaron las mismas métricas con el fin de comparar los resultados correspondientes.

10.- Caso de prueba: El caso de prueba propuesto es el reconocimiento de actividades humanas. Es una tarea de clasificación multiclase y cuenta con alta dimensionalidad. Además, existe la base de datos pública "UP-Fall Detection Dataset" [29] sobre la que el algoritmo original ya ha sido probado.

11.- Exploración de los datos y métricas de caso de estudio: Para lograr entender de mejor forma los datos de HAR, se realizó una exploración preliminar de la base de datos "UP-Fall Detection Dataset" [29], buscando desbalances, datos y categorías por limpiar. Se proponen métricas específicas para el caso de estudio, tiempo de ejecución, precisión, exactitud y puntuación F1.

12.- Conclusiones y trabajo futuro: Según los resultados anteriores, se ofrecen las conclusiones originadas del análisis comparativo del modelo sin modificaciones. Fue propuesto también el trabajo futuro del presente proyecto, que buscará extender las capacidades del algoritmo y la implementación.

En el capítulo 2 se discuten los antecedentes del modelo de redes de hidrocarburos artificiales, el marco teórico sobre el aprendizaje automático y su relación con la optimización, técnicas de escalamiento y procesamiento paralelo; finalmente se discuten las características más relevantes de la tarea de reconocimiento de actividades humanas. Posteriormente, el capítulo 3 describe la construcción y pruebas de las modificaciones propuestas. Mientras que los resultados obtenidos y su discusión están contenidos en el capítulo 4. Por último, en los capítulos 5 y 6 se ofrecen las conclusiones encontradas y las propuestas de trabajo futuro respectivamente.

Capítulo 2

Antecedentes y estado de la cuestión

El Capítulo 2 presenta el estado actual del modelo de Redes de Hidrocarburos artificiales, los algoritmos de optimización y su contacto con aplicaciones de reconocimiento de actividades humanas. Esta sección del documento sirve como punto de partida para entender el problema de HAR y su relación con el aprendizaje automático en la actualidad.

En este capítulo se discutirán algunos conceptos principales para los algoritmos y modelos presentados; por lo que es importante explicarlos brevemente. La función objetivo es en realidad la medida cuantitativa del funcionamiento del sistema que se desea optimizar (maximizar o minimizar) [46]. La función de actualización es la estrategia por la cual se recalcula la posición de los agentes en el espacio de búsqueda. Por último el espacio de búsqueda es el conjunto de soluciones posibles para una función objetivo [46].

2.1. El modelo de Redes de Hidrocarburos Artificiales

Las redes de hidrocarburos artificiales son un modelo de aprendizaje automatizado derivado de las de redes orgánicas artificiales [42][41]. Este modelo de redes orgánicas artificiales parte del supuesto que los compuestos orgánicos buscan minimizar la energía química en las estructuras moleculares formadas.

Las ventajas observadas del modelo de redes orgánicas artificiales frente a otros modelos de redes, como las redes neuronales artificiales, son la estabilidad algorítmica, el encapsulado y herencia de información y el entendimiento parcial de los datos a través de las estructuras obtenidas. Las redes de hidrocarburos artificiales también han mostrado buenos resultados en problemas y aplicaciones con datos multivariados, no lineales y con incertidumbre [40]. Actualmente el modelo de AHN ha sido entrenado en conjunto con un algoritmo basado en el gradiente. Los algoritmos de entrenamiento que se incorporan a los modelos de aprendizaje automático son los responsables de ajustar los parámetros internos del modelo, alcanzando los mejores resultados para la tarea designada. Este proceso es en sí mismo una tarea de optimización que se reduce a encontrar la mejor solución para minimizar o maximizar la función objetivo; en la que la solución es un conjunto de pesos o variables internas del modelo de aprendizaje automático y la función objetivo es la métrica de error seleccionada para evaluación. El modelo de AHN [42] en realidad cuenta con dos tipos de variables, la posición de las

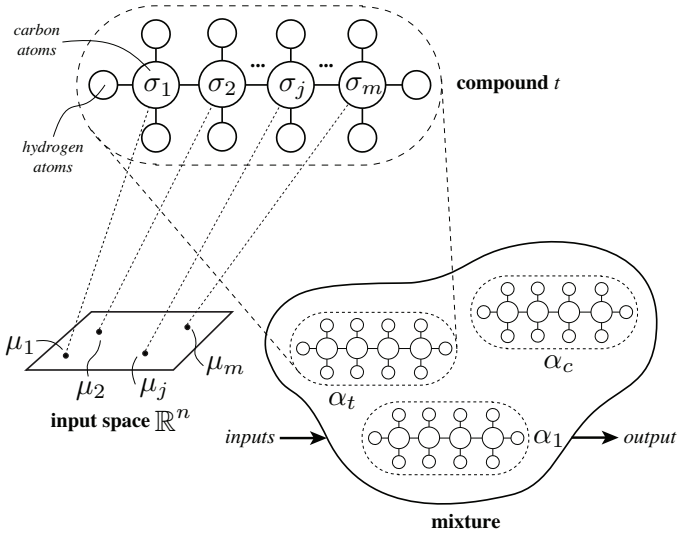

FIGURA 2.1: Formación de una molécula, un compuesto y una mezcla en el modelo AHN [41]

moléculas que forman el compuesto y los valores de los elementos hidrógeno y carbono que forman cada molécula. La figura 2.1 fue tomada del artículo [41] y muestra los valores de los hidrógenos y carbonos en cada elemento del compuesto t y los centros o localizaciones de las moléculas en μ_m. Esta tesis se enfoca en encontrar las posiciones de las moléculas en el espacio $\mathbb{R}^n$ mostrado en la figura 2.1. Si se buscara encontrar los valores de los elementos y la localización de las moléculas, el problema de optimización se convertiría en un problema multi-objetivo [9]. Por esta razón, el entrenamiento será enfocado en un único objetivo.

Aunque las redes de hidrocarburos artificiales han presentado resultados favorables en aplicaciones de clasificación y regresión lineales y no lineales frente a otros modelos entrenadas con el algoritmo de propagación hacia atrás [40][42], este modelo presenta inconsistencias de precisión y tiempo de entrenamiento, especialmente para funciones con gran número de dimensiones y con conjuntos de datos grandes. Estas dificultades se presentan ya que el algoritmo de entrenamiento actual es un algoritmo determinista y no uno estocástico, alentando el proceso de entrenamiento ante gran cantidad de datos y aumentos en la dimensionalidad del problema. Por esta razón, la propuesta presentada busca aportar una solución para mejorar ambos aspectos, mediante la aplicación de un nuevo algoritmo metaheurístico y su comparación con el modelo AHN modificado con optimizadores metaheurísticos como el optimizador de lobos grises, optimizador de murciélagos y de parvada.

2.2. Métodos de optimización

Snyman define de forma general un problema de optimización como el estudio para determinar la mejor solución para un problema o modelo definido matemáticamente [53]. Este problema puede, o no, estar acotado por límites iniciales que reducen las soluciones posibles. Formalmente se define como [53] la formulación y solución de un problema

(de maximización o minimización) de la forma matemática general expresada en (2.1) y sujeto a condiciones $g_j(x)$ y $h_j(x)$ de la forma enunciada en (2.2) y (2.3).

$$Max|Min, f(x), x = [x_1, x_2, \cdots, x_n]^T \in \mathbb{R}^n \tag{2.1}$$

$$g_j(x) \leq 0, j = 1, 2, \cdots, m \tag{2.2}$$

$$h_j(x) = 0, j = 1, 2, \cdots, r \tag{2.3}$$

Los elementos de x en la Ecuación 2.1 son las dimensiones o variables de diseño del problema, $f(x)$ es la función objetivo y las funciones $g_j(x)$ y $h_j(x)$ son llamadas restricciones de desigualdad y restricciones de igualdad respectivamente.

Existen diferentes formas de resolver un problema de optimización, llamadas métodos de optimización. Los métodos de optimización pueden clasificarse en dos principales tipos: los métodos clásicos y los métodos metaheurísticos. Los métodos clásicos incluyen los métodos lineales, no lineales, estocásticos y dinámicos. Estos métodos clásicos son, en general, métodos matemáticos deterministas o exactos que se concentran en encontrar el valor óptimo, máximo o mínimo de la función objetivo [46]. En contraste, los algoritmos metaheurísticos están centrados alrededor de encontrar una solución suficientemente buena y de forma menos costosa computacionalmente según un criterio de aceptación basado en una función objetivo, es decir, aproximan soluciones óptimas.

Los problemas de optimización pueden o no contar con incertidumbre, es decir que todos los estados de todas las variables del ambiente son conocidas en todo momento o no. Yu Ermoliev habla de esta división en el manejo de la incertidumbre como aquella característica particular de la optimización con métodos estocásticos [12]. Explica también cómo los algoritmos deterministas son especialmente insensibles a circunstancias externas que pueden afectar al ambiente del problema ya que no se consideran como variables internas y requieren de un estudio por experiencia para filtrar soluciones imposibles en la realidad. [19] Los algoritmos estocásticos integran esta incertidumbre al mantener variables indeterminadas en sus procesos de optimización y buscando encontrar una solución para los valores más probables que estas variables indeterminadas podrían tomar.

Incluidos en los métodos clásicos de optimización están los métodos iterativos [24], en los que existen algunos algoritmos famosos como los métodos de mínimos cuadrados [10], gradientes y el método de Newton. Estos algoritmos se basan en la aproximación a una solución tomando en cuenta los estados anteriores; no haciendo cambios aleatorios sino evaluando su acercamiento progresivo a la solución según sus resultados anteriores. Los algoritmos iterativos guardan algunas similitudes de operación con los algoritmos metaheurísticos, que al ser de principal interés para el trabajo presentado, se discuten en una sección reservada a continuación.

2.2.1. Algoritmos de optimización metaheurística

Los algoritmos basados en el gradiente utilizan información obtenida de las derivadas de varios órdenes. Esta característica los vuelve muy buenos para problemas donde se busque optimizar para mínimos o máximos locales. En su mayoría pueden ser considerados algoritmos glotones en el caso de presentarse un problema no convexo, ya que la existencia de máximos o mínimos locales provoca que estos algoritmos se queden atrapados en ellos en vez de encontrar la solución óptima global. Otra limitante es que, al basarse en derivadas y gradientes, es necesario que exista la derivada de la función objetivo del orden que el algoritmo utilice. Existe también el problema de ser necesaria la continuidad de la función ya que de no serlo no existirá derivada en los puntos de discontinuidad.

Al presentarse las limitantes anteriores, muchos modelos de aprendizaje automático han recurrido a utilizar nuevos algoritmos de entrenamiento. El caso más notable y relacionado con la aplicación discutida es el entrenamiento del modelo de redes neuronales, que ha sido modificado por David Montana para incorporar algoritmos genéticos, el algoritmo de parvada por V.G. Gudise [18], el algoritmo de optimización inspirado en el método de caza del lobo gris por Seyedali Mirjalili [33] [32] [34]. En todos los casos se ha observado que las capacidades de precisión se han mantenido o mejoran a la par de mejorar la eficiencia para encontrar una solución aproximada para la función objetivo; para el caso del entrenamiento se refiere a la función de error.

Algoritmos genéticos - GA

Los algoritmos genéticos modelan la recombinación genética entre elementos; haciendo un símil entre arreglos de variables que constituyen una solución de la función objetivo [52]. Estos arreglos son evaluados y organizados según su desempeño al minimizar o maximizar. Los elementos con mejores resultados son seleccionados y posteriormente sometidos a un proceso de recombinación o cruza, poblando un nuevo conjunto de soluciones. Este nuevo conjunto se compone de combinaciones de los elementos que conformaban a sus padres además de mutaciones introducidas por un criterio probabilístico establecido previamente. El proceso se repite tantas generaciones especificadas o hasta que se alcance una solución que cumpla con los parámetros deseados [20][50][11].

Algoritmo de optimización por parvada - PSO

El algoritmo de parvada busca modelar el sistema de navegación y ordenamiento de las parvadas cuando se mueven por una trayectoria [50][52]. Este algoritmo explora el espacio de una función ajustando las trayectorias de los agentes, llamados partículas. El movimiento de cada partícula es modelado como un vector posicional dependiente del tiempo. Esto provoca que el modelo pueda tener en cuenta la velocidad de movimiento de cada partícula, así como compartir la información de movimiento entre agentes. Cuando una partícula encuentra un punto solución que es mejor que la mejor solución hasta ahora encontrada por la parvada, no solo continua moviéndose hacia esa dirección, también actualiza un parámetro de "inercia" para toda la parvada, provocando que todas las partículas comiencen a explorar soluciones dirigidas hacia ese punto al ser movidas hacia él. Esto no lo vuelve vulnerable a mínimos globales ya que la inercia

TABLA 2.1: tabla comparativa de características de interés en algoritmos metaheurísticos

Algoritmo	Parámetros ajustables	Incentivo a exploración	Castigo de sobre-explotación	Funciones de movimiento	Control sobre puntos de interés
PSO	6	No	No	1	No
GWO	1	No	No	3	No
BA	5	No	No	1	No

no es el único parámetro para dictar la actualización del vector de movimiento, también se toma en cuenta el sentido y dirección propios de la partícula [60][11][38].

Algoritmo de caza del lobo gris – GWO

El algoritmo de caza del lobo gris u optimizador del lobo gris, busca modelar el comportamiento de una manada de lobos al cazar. Específicamente inspirado en la dinámica social que existe entre los integrantes de la manada, contemplando a un lobo alfa, un lobo beta, un lobo delta, en orden jerárquico. En el algoritmo los lobos que toman estas posiciones de poder son las soluciones encontradas que tengan el valor mínimo o máximo global encontrado en cada iteración [34][33][32]. Existe una tendencia a que los demás lobos sigan en primer lugar al alfa, después al beta y por ultimo al delta, así como un parámetro de aleatoriedad que le permite al algoritmo no caer en un mínimo local constantemente [50][11][27].

Algoritmo de murciélago - BA / BAT

Es un algoritmo de optimización inspirado en los sistemas de ecolocación que utilizan algunas especies de murciélagos para la navegación [66] [64]. El algoritmo permite el ajuste de comportamiento de los murciélagos según la intensidad de la señal (loudness), el ancho de onda y la frecuencia. El algoritmo inicializa a los murciélagos con una velocidad y posición aleatoria en el espacio; progresivamente actualiza la posición de los murciélagos recalculando la velocidad específica de cada uno. Esta velocidad es producto de la frecuencia multiplicada por la longitud de onda. Posteriormente se actualiza el volumen del "ruido" emitido por cada murciélago según su resultado en la evaluación [17] [65].

2.2.2. Ventajas y desventajas

Los algoritmos metaheurísticos discutidos anteriormente han sido utilizados para procesos de entrenamiento de modelos de aprendizaje automático; especialmente los algoritmos PSO y, más recientemente, el algoritmo GWO. Estos algoritmos presentan ventajas y desventajas dependiendo del problema, dimensiones y parámetros de ajuste [62]. Es muy importante recordar que no existe un algoritmo mejor que otro en el absoluto de aplicaciones, simplemente existen mejores aplicaciones para cada uno de ellos. El algoritmo propuesto en este trabajo cuenta con algunas características inspiradas de otros métodos populares. Estas características se ven resumidas en la tabla 2.1.

De la tabla 2.1 es importante resaltar que la cantidad de parámetros que un algoritmo metaheurístico requiere impacta directamente en la complejidad de su uso. Los algoritmos metaheurísticos dependen completamente de estos parámetros iniciales para configurar la exploración, su velocidad de convergencia e incluso los recursos computacionales consumidos. Por esta razón, se busca que el algoritmo propuesto tenga la menor cantidad de parámetros ajustables sin sacrificar flexibilidad, de esta forma se facilita su aplicación y ajuste para múltiples tipos de problemas.

Otra de las características centrales es que no todos los algoritmos metaheurísticos contemplan lógica que incentive la exploración de la función en su totalidad. Incorporar cambios aleatorios en el conjunto de soluciones no es incentivar la exploración, ya que esto solo introduce aleatoriedad que espera evitar que las soluciones caigan en mínimos o máximos locales.

2.2.3. Selección de un método de optimización

Todos los algoritmos metaheurísticos para optimización son aplicables a una gran cantidad de problemas posibles, por esto se corre el riesgo de pensar que existe o existirá un solo algoritmo capaz de presentar los mejores resultados en cualquier problema, pero esto no es posible. Lo anterior está sustentado en el teorema de No-Free-Lunch [62], que ha probado lógicamente que no existe un solo algoritmo metaheurístico capaz de dar los mejores resultados en cualquier naturaleza del problema. Por esta razón, la correcta selección del algoritmo de optimización cobra gran importancia.

Existen muchos factores a considerar cuando se busca seleccionar un algoritmo de optimización, factores que comúnmente limitan las opciones y es importante tenerlos en cuenta. El algoritmo seleccionado para cierta tarea dependerá en buena medida del tipo de problema, la naturaleza del algoritmo, la calidad deseada del resultado, los recursos computacionales disponibles y la disponibilidad de la implementación. Entonces, el reto de la selección de un algoritmo de optimización computacional consiste en encontrar el algoritmo más adecuado para un problema particular, con el fin de obtener una solución suficientemente buena de acuerdo a las especificaciones deseadas, en un tiempo tolerable y con recursos limitados.

Es necesario evaluar si la función objetivo es lineal, su dimensionalidad, continuidad y el tipo de solución que se busca. Por ejemplo, si se busca un mínimo o máximo global, no es posible solucionar el problema con algoritmos que encuentren mínimos o máximos locales de la función objetivo [19]. En caso de la continuidad, no se podría usar un algoritmo que utilice derivadas o gradientes para solucionar un problema discontinuo o discreto.

Tomar en cuenta los recursos disponibles también es necesario para la selección. Los recursos limitados a considerar son tiempo de ejecución, capacidad computacional y la disponibilidad de la implementación del algoritmo [3]. Entonces se buscará ponderar los recursos invertidos contra las especificaciones de la solución requerida; hay que recordar que no en todos los casos es necesaria la mejor solución, solamente es necesaria una solución suficientemente buena de acuerdo a las especificaciones de precisión, confianza y rapidez de ejecución [11].

De acuerdo a lo discutido en esta sección, es claro que para solucionar problemas de optimización se requiere de un profundo conocimiento de las condiciones y características propias del problema en cuestión; pero también es claro que el conocimiento general de los algoritmos más utilizados y sus aplicaciones en diferentes campos es igualmente necesario.

2.3. Escalabilidad del aprendizaje automático

Bekkerman enuncia seis características principales de un problema que hacen necesaria la aplicación de técnicas de escalamiento para el aprendizaje distribuido [3]. Dado el estado actual de la implementación del algoritmo de AHN, este trabajo se concentra en tres de las características enunciadas.

1. **Gran cantidad de observaciones**: Referente a la cantidad de observaciones que conforman a la base de datos de entrenamiento o de pruebas. Al contar con mayor cantidad de puntos a resolver, aumenta el tiempo y capacidad de procesamiento que se requiere para encontrar una solución.
2. **Gran cantidad de dimensiones de entrada**: La cantidad de variables o dimensiones que forman a cada observación. El aumento dimensional en una tarea especifica significa que encontrar una solución exacta es cada vez más complejo, impactando directamente sobre el tiempo de entrenamiento. Además de incrementar la inestabilidad del sistema ya que cada vez es más probable que el algoritmo implementado no logre encontrar la solución exacta al problema.
3. **Complejidad del modelo y del algoritmo**: Algunos modelos de aprendizaje funcionan a través de algoritmos complejos o con subrutinas computacionalmente costosas como ordenar arreglos o un límite mínimo de precisión.

En la actualidad las aplicaciones del aprendizaje automatizado, tanto supervisado como no supervisado, se ha extendido a fenómenos monitorizados que generan millones de datos diariamente. Además del gran volumen de datos, las aplicaciones actuales han llegado al orden de 10^6 dimensiones. Estas nuevas características presentan un gran reto para la aplicación de los modelos de aprendizaje automático en sistemas locales con una agenda de procesamiento serializada [3].

2.3.1. Procesamiento paralelo y cómputo distribuido

En lo que respecta al volumen de datos, se han desarrollado estrategias que combinan la capacidad de almacenar grandes estructuras en agrupaciones de computadoras, además de facilitar su procesamiento a través de técnicas de distribución. Esta es la estrategia preferida para procesar conjuntos grandes de datos, distribuir secciones del conjunto entre los miembros de una agrupación de computadoras para aprovechar sus recursos de almacenamiento, ancho de banda y capacidad de procesamiento. Han surgido muchas herramientas que facilitan estas distribuciones entre las que destaca Apache Spark.

Apache Spark es un marco de software que unifica múltiples tareas como la ejecución de funciones de preparación y extracción, ejecución de modelos y consultas a conjuntos de datos. Spark distribuye las tareas de procesamiento entre múltiples ejecutores que se

comunican a un gestor principal, dejándole saber el estado de las tareas asignadas, los recursos disponibles con los que cuenta y los resultados al terminar [16]. Además, Spark extiende capacidades de permanencia o resiliencia de datos al gestionar automáticamente copias de extractos del conjunto original entre múltiples ejecutores, asegurando un riesgo mínimo de pérdida de información [49][30].

La cantidad de dimensiones es un problema cuando se trata de un modelo o algoritmo iterativo entre cada característica [3]. Tal es el caso de los algoritmos metaheurísticos que reajustan cada dimensión de sus agentes de forma iterativa. Esto significa que a mayor cantidad de dimensiones, mayor cantidad de iteraciones de ajuste para cada agente del algoritmo. Una de las formas más efectivas para disminuir el tiempo efectivo que toma realizar estas iteraciones es la paralelización del procesamiento. Es importante decir que paralelizar el procesamiento y distribuirlo son dos técnicas fundamentalmente distintas. A diferencia de la distribución que realiza Apache Spark [49], discutida anteriormente, la paralelización se realiza en el hardware de cada máquina, asignando una operación a cada núcleo de un computador. Estas operaciones son sencillas y se realizan rápidamente y al mismo tiempo entre los núcleos, posteriormente se gestionan los resultados para minimizar el tiempo de espera de cada núcleo.

El problema discutido anteriormente está relacionado también con la complejidad algorítmica, y ambos han sido atacados con el mismo acercamiento. La complejidad algorítmica se origina al utilizar modelos con representaciones no lineales [3]. Aunque el tratamiento de datos, así como un análisis exploratorio que determine características claves de los datos pueden facilitar el aprendizaje automático y simplificar la arquitectura de los modelos o simplemente solucionarse con uno más sencillo, actualmente se busca que el aprendizaje automático sea lo menos manual posible. Esto empuja desarrollos que requieran menor intervención o pre-procesamiento de datos, es decir modelos más complejos que lidien con problemas como la no linealidad. Debido a la relación entre los modelos complejos, la naturaleza iterativa de estos, así como las aplicaciones con gran dimensionalidad, la estrategia de paralelización es utilizada para solucionar ambos problemas.

Se han desarrollado tecnologías que permiten paralelizar procesos en unidades de procesamiento gráfico, que dentro de su configuración física, cuentan con múltiples núcleos de bajo procesamiento pero muy veloces [23]. La cantidad de núcleos presentes en una unidad de procesamiento gráfico supera por mucho la cantidad de núcleos en una uni- dad de procesamiento central; aunque no tienen la misma velocidad o capacidad de procesamiento, son altamente efectivos para operaciones sencillas. La tecnología más utilizada actualmente para realizar la paralelización de procesamiento en una unidad de procesamiento de gráficos es CUDA, tecnología desarrollada por NVIDIA para sus tarjetas gráficas [35] [23]. El nombre CUDA está compuesto por las siglas del nombre completo en inglés “Compute Unified Device Architecture”. Esta tecnología tiene implementaciones para los lenguajes de desarrollo de aprendizaje automático más utilizados, así como amplia documentación.

CUDA ha sido utilizado en estudios y desarrollos de minería de datos para la implementación de algoritmos paralelizados a escala [22], clasificación de proteínas en biología computacional [1] e incluso modelos de comportamiento de contaminantes en el ambiente y sus efectos [36]. La tecnología desarrollada por NVIDIA en 2001 [47] fue

la piedra angular para el desarrollo de aplicaciones paralelizadas en unidades gráficas; la meta era modelar los problemas como tareas de procesamiento gráfico tradicionales, poco a poco la necesidad de un acercamiento más general para el aprovechamiento de las GPUs se hizo mayor, culminando en la liberación de CUDA en 2006 como una plataforma para el desarrollo de tareas de software que podrían beneficiarse de la paralelización a escala. CUDA en realidad consiste en la liberación del acceso a las unidades de aritmética (ALU - Arithmetic Logic Unit), la integración de un conjunto de instrucciones de propósito general en lugar de uno limitado a computo gráfico y el libre acceso a lectura y escritura en memoria de todas las unidades ejecutoras en la GPU [47].

2.4. Reconocimiento de actividades humanas - HAR

Tener la capacidad de reconocer actividades humanas ha sido un tema de investigación abierto en los últimos 20 años. Esto se debe al impacto del reconocimiento de actividades humanas o HAR, por sus siglas en inglés, en aplicaciones médicas, de seguridad, entretenimiento e incluso militares. Respecto a aplicaciones médicas, muchos pacientes con distintas enfermedades, requieren de seguir estrictamente algunas rutinas, evitar realizar algunas actividades e incluso hacer ejercicios específicos [26]. HAR es importante para el seguimiento del comportamiento y tratamiento de un paciente, ayudando al diagnóstico médico y su prognosis. En cuanto a seguridad y aplicaciones militares, el seguimiento de soldados, su posición, actividades y estado de salud y seguridad es vital en algunos operativos.

Dada la cantidad de actividades a clasificar, así como las similitudes entre ellas, el reconocimiento mediante métodos deterministas y clásicos se complica porque existen demasiadas combinaciones posibles de características y actividades. Por esta razón se han incorporado técnicas de aprendizaje automático. El motivo de incorporar el aprendizaje automático es que la mayoría de las aplicaciones requieren de la construcción e identificación de patrones en los datos recogidos para describir, analizar y predecir datos [26] [40]. Es importante considerar que las actividades a identificar se realizan en una serie de tiempo en el que se pueden realizar múltiples actividades a la vez; esta consideración solo puede ser resuelta a través del aprendizaje automatizado y es intrínseca a la definición del problema de HAR, por lo que el tamaño de la ventana de datos es crucial para el problema.

2.4.1. Aprendizaje automático en el Reconocimiento de Actividades Humanas

El aprendizaje automático ha sido aplicado para el reconocimiento de actividades humanas en los últimos años dada su capacidad de aprender de grandes volúmenes de datos y la flexibilidad de aprender patrones desde imágenes y sensores discretos que aplican para distintos sujetos en situaciones no estáticas [56][58]. Se han utilizado modelos de redes neuronales [4], maquinas de soporte vectorial [6], arboles de decisión [13], bosques aleatorios [57] y lógica difusa.

Una de las mayores dificultades a las que se enfrenta HAR es la diferencia entre el entrenamiento con datos recogidos en condiciones controladas y las condiciones del mundo real. Como presenta Turaga [58], uno de los mayores problemas es la gran cantidad de

datos generados para entrenar modelos en HAR, lo que obliga a los sistemas a lidiar con vídeos e imágenes de baja resolución. Otro punto importante que se discute en [58] es la variabilidad de una misma actividad entre sujetos dadas distintas condiciones antropométricas. Además el autor realiza una descripción del flujo de un clasificador de actividades que lidia con secuencias de imágenes; pasando de la entrada de cuadros a la extracción de características primitivas, seguido de una descripción de media complejidad de una actividad concreta y culminando con la descripción semántica de una actividad compleja.

También se han realizado ejercicios de clasificación utilizando datos de acelerómetros embebidos en teléfonos celulares, que mediante un profundo análisis de componentes principales, lograron mostrar el comportamiento y la importancia de variables y atributos basados en la frecuencia y no solo en tiempo [56]. Este trabajo mostró cómo la aplicación de una ventana de tiempo puede afectar radicalmente los resultados de exactitud de un clasificador y el impacto que tiene la reducción dimensional del problema para mejorar drásticamente los resultados de la clasificación. Pero no siempre es posible reducir las dimensiones significativamente, ya que al aumentar la cantidad de actividades a clasificar, necesariamente se generan más dimensiones que las caractericen. Esto sustenta la necesidad de un desarrollo que soporte aprendizaje de bases de datos con un mayor número de dimensiones.

Es importante resaltar que existen desarrollos que involucran sensores externos, otros sensores vestibles o un conjunto de datos híbridos entre ambos. Trabajos como el presentado por Cheng [8] muestran el uso de sensores vestibles, que a diferencia de acelerómetros embebidos [28] en un dispositivo móvil [67], permiten lecturas precisas de distintas extremidades; abriendo la posibilidad a un análisis especifico de características que son imposibles de ser observadas con cámaras. La importancia de este trabajo radica en el análisis de nuevas posibilidades para la recolección de datos y el impacto que puede lograrse al entrenar clasificadores con fuentes de datos internas y externas [51].

El modelo de AHN ha sido utilizado con el fin de atacar el problema de flexibilidad en HAR, refiriéndose a distintos aspectos de flexibilidad del sistema, incluyendo nuevos sujetos o reconocimiento de nuevas actividades, sugiriendo que un clasificador debe ser capaz de lidiar con varios sujetos realizando múltiples actividades en días distintos [42][43]. El modelo de AHN ha sido probado para clasificación, donde se prueba su efectividad para el reconocimiento de actividades físicas, tolerancia a ruido en los sensores de recolección de datos y que logra un reconocimiento robusto en comparación a otros métodos de aprendizaje automático [40]. Estos experimentos se realizaron sobre una misma base de datos, en la que se realizaron tres pruebas. La primera prueba hecha con los datos originales, la segunda con características reducidas utilizando el método de eliminación recursiva de características, por último la tercera prueba se realizó introduciendo 1 %, 15 % y 30 % de ruido a los datos en forma de entradas aleatorias de datos.

El presente desarrollo busca aportar una solución para mantener o mejorar los resultados de precisión, tiempo de ejecución y exactitud del modelo actual de redes de hidrocarburos artificiales mediante la incorporación de un nuevo algoritmo metaheurístico

como algoritmo de entrenamiento. A la par de la implementación de una estrategia de paralelización de procesamiento para reducir el tiempo efectivo de ejecución.

El algoritmo no solo debe servir para el entrenamiento en el estado actual, se busca que en el futuro pueda ser modificado para lograr la optimización de la cantidad de moléculas necesarias para formar el compuesto utilizado por el modelo, posicionamiento de las moléculas y distancia entre elementos; es decir que el algoritmo pueda ser modificado para lograr una optimización multiobjetivo para la estructura AHN.

Capítulo 3

Descripción de la propuesta

En este capítulo se presenta la propuesta de trabajo, detallando los cambios realizados así como los fundamentos matemáticos del diseño del algoritmo de entrenamiento para el modelo de AHN. Primero se realizará una comparación donde se contrasta el estado actual de la implementación contra el sistema propuesto; posteriormente la inspiración del algoritmo diseñado y su relación con las características específicas de los algoritmos de optimización. Esta sección también presenta el pseudocódigo del algoritmo, métricas de soporte y una descripción breve del ambiente de pruebas utilizado.

En la figura 3.1 se muestra el funcionamiento actual de la implementación de AHN para la clasificación de actividades humanas. En esta figura se muestra cómo la base de datos es utilizada por el modelo AHN, que es entrenado al utilizar el algoritmo basado en el gradiente. También se puede notar cómo la implementación actual se apoya en un solo núcleo de procesamiento para el entrenamiento. Este entrenamiento es en realidad el proceso de optimización o minimización del error medio.

En la figura 3.2 se muestran varias diferencias respecto a la figura 3.1. Estas diferencias son los cambios propuestos en este trabajo, que incluyen el proceso de balanceo de clases para aumentar la representación de las minorías en el conjunto de entrenamiento, utilizando la técnica de sobremuestreo sintético de minorías (SMOTE), la implementación de un algoritmo metaheurístico que optimice la función objetivo, que es la función de error. También incorpora múltiples núcleos marcados con la letra n, que serán los encargados de procesar las operaciones más básicas según una división de tareas por colas para lograr la aproximación de una solución; esto es la estrategia de paralelización de procesamiento.

Tal como se explicó en la Sección 2.1, este trabajo buscará encontrar los centros μ_n de las moléculas del modelo AHN utilizando algoritmos de optimización metaheurística. De acuerdo a la definición formal del problema de optimización en la Sección 2.2, se busca minimizar la función objetivo. Donde la función objetivo será el error cuadrático medio del modelo AHN, evaluándolo en una solución posible. En la Ecuación 3.1 se enuncia el problema de optimización incluyendo los valores de hidrógenos y carbonos nH y nC. Como se explica en la Sección 2.1, los valores de hidrógenos y carbonos en el modelo son dados (obtenidos en cada iteración mediante el algoritmo de mínimos cuadrados [41]) y no son variables de diseño en el alcance de esta investigación.

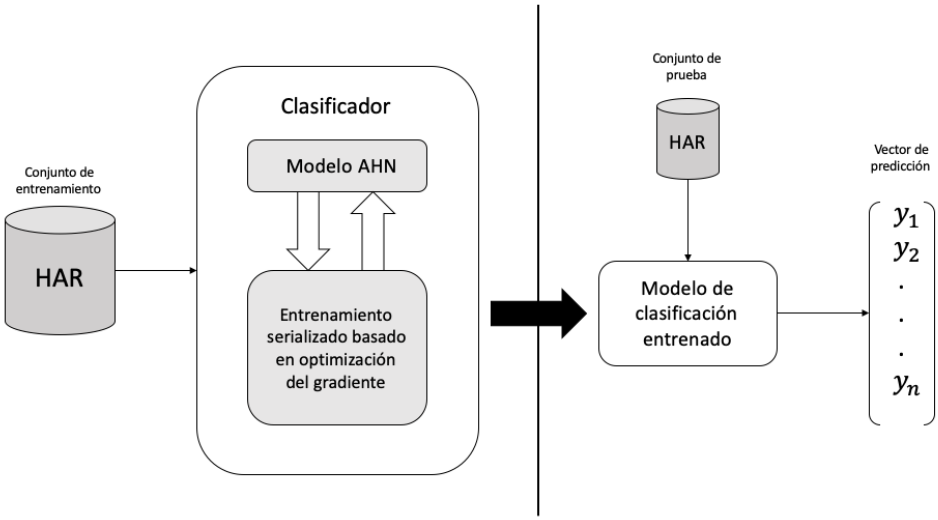

FIGURA 3.1: Diagrama de la implementación actual

$$\begin{aligned} Min, AHN(r, nH, nC), r = [r1, r2, \cdots, r_i] \in \mathbb{R}^i, \\ nH = [H_1, H_2, \cdots, H_i] \in \mathbb{R}^{2(i+1)}, \\ nC = [C_1, C_2, \cdots, C_i] \in \mathbb{R}^i \end{aligned} \tag{3.1}$$

En la Ecuación 3.1 se define r como una solución posible propuesta por un algoritmo de optimización que representa los centros de las moléculas.

En ambos casos, una vez finalizado el proceso de entrenamiento, el modelo es exportado y puesto a prueba utilizando un conjunto de datos no incluido en el conjunto de entrenamiento. Este conjunto de datos de prueba es una muestra aleatoria de la base de datos inicial. El resultado de este proceso de clasificación es en realidad un vector donde cada elemento es en realidad otro vector que contiene la probabilidad de pertenencia de una observación a cada una de las 11 clases disponibles. La clase con la mayor probabilidad de pertenencia es seleccionada como la clasificación final y es representada en el vector de resultados como y_n.

La escalabilidad del modelo actual de AHN se ve limitada, dado el algoritmo de entrenamiento basado en el gradiente que no presenta buenos resultados de acuerdo al tiempo de entrenamiento y precisión ante problemas con gran número de dimensiones y registros.

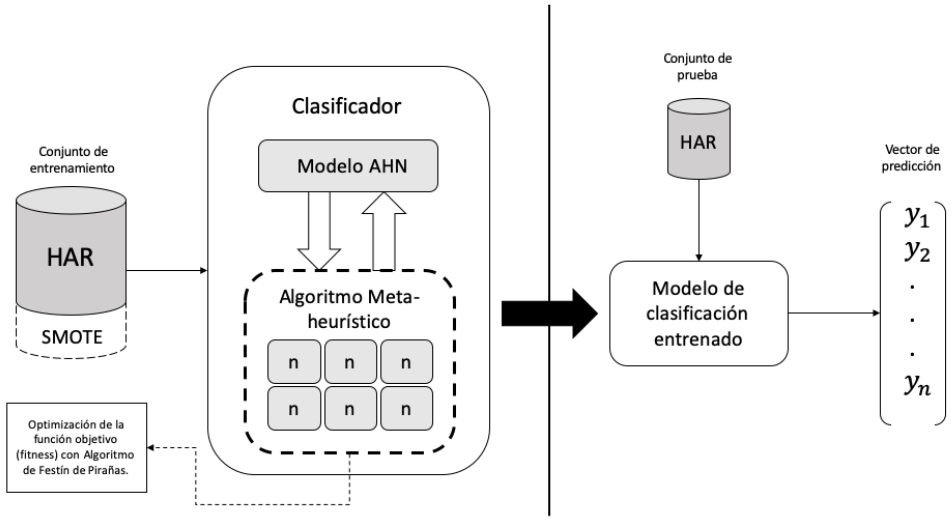

FIGURA 3.2: Diagrama de cambios propuestos

3.1. Algoritmo de festín de pirañas - PFO

3.1.1. Inspiración

El algoritmo estará inspirado en las dinámicas sociales de esta especie de pirañas durante el periodo de caza. Esta especie fue seleccionada por sus hábitos alimenticios que tal como explica Carvalho [5], destaca de entre otras especies por el alto porcentaje de carne de peces y otros vertebrados en su dieta. Este porcentaje se encuentra entre 44.1 % y 56.3 % dependiendo de si se encuentran en ríos o lagos respectivamente. Además, esta especie ha mostrado un comportamiento social peculiar gracias a su capacidad de comunicación a través de ondas sonoras. Se ha observado que existen distintas ondas que permiten a esta especie comunicarse; indicando distintas actividades con diferente frecuencia de onda y duración [31]. Específicamente, se han encontrado diferencias entre las ondas utilizadas para indicar eventos como circundar a una potencial presa y durante la pelea, así como el momento de persecución de la presa [31] [55].

Este fenómeno es interesante porque, durante la caza, el cardumen se esparce para encontrar presas potenciales. Cuando uno de los sujetos encuentra una presa, emite una señal a los demás elementos del cardumen para que comiencen a acercarse a la zona. Si la presa es demasiado grande para ser consumida de un bocado, se ha observado que el cardumen busca inmovilizarla tomando bocados grandes de la cola, además provocando fuerte sangrado que la debilita. [48][31] [15].

Las pirañas no solo siguen las señales de los demás, también se guían por la presencia de presas cercanas a su radio de ataque. La caza se da de forma ordenada, con solamente uno o dos sujetos alimentándose a la vez de la misma presa y moviéndose a buscar más en los alrededores. Adicionalmente, si otro integrante encuentra otra presa al alcance del

TABLA 3.1: Tabla de características del algoritmo PFO

Caracteristica general	**Caracteristica en PFO**
Variable de decisión	Posición de las pirañas en cada dimensión
Solución	Vector de posición de cada piraña en el espacio explorado
Solución anterior	Posiciones anteriores de las pirañas en el espacio
Nueva solución	Nueva posición de cada piraña en el espacio de búsqueda
Mejor Solución	Posición de la presa más grande encontrada durante la búsqueda
Función objetivo	Tamaño de la presa
Solución inicial	Posicionamiento aleatorio de de las pirañas en el espacio
Proceso de generación de nuevas soluciones	Redireccionamiento de navegación individual a puntos de interés global e individual en cada dimensión con selección de ruleta.

cardumen, también lanza una alerta que invita a los miembros cercanos a concentrarse en la zona. Las alertas se dan directamente en forma de ondas sonoras [55] [45].

La dinámica social de esta especie de piraña, en el momento de caza, es muy similar a un proceso de optimización metaheurística de una función objetivo. Además, el fenómeno presenta la posibilidad de incorporar comunicación entre los agentes y aleatoriedad durante el movimiento individual y grupal, previniendo el estancamiento en mínimos o máximos locales. Esto facilita el modelado del fenómeno natural y su traducción a un algoritmo por la posibilidad de hallar símiles directos al proceso de optimización.

3.1.2. Diseño del algoritmo

En esta sección se discuten similitudes entre el fenómeno observado y su modelado. También se explica el flujo detallado del algoritmo. A continuación se presenta una lista con las características que describen el algoritmo; posteriormente se discute cada una y se ofrecen las postulaciones correspondientes:

En el algoritmo se toman los integrantes del cardumen, es decir las pirañas, como los sujetos de la población. La posible solución se representa en la posición en el espacio de soluciones de las pirañas. La mejor solución será la posición de la presa más grande.

Los criterios de actualización de los sujetos se dividen en dos: el criterio individual y el criterio social. Como se ha discutido anteriormente, durante los festines, las pirañas pueden guiarse por la presencia de una presa cercana, este es el modelo para el criterio individual. En contraste, el criterio social está inspirado en las señales sonoras que emiten las pirañas que encuentran una presa; estas señales son escuchadas por los demás integrantes del cardumen y estos priorizan su movimiento hacia las zonas identificadas.

En el algoritmo se permitirá la existencia de las presas; las presas son las posiciones que registren el valor mínimo o máximo de la función objetivo según la tarea de optimización. Cada presa tiene una cantidad finita de "alimento" que disminuye con cada acercamiento de una nueva piraña a esa posición. Esta cantidad de "alimento" es el radio de ataque de cada presa, el cual se reduce en cada iteración.

El algoritmo propuesto jerarquiza a los individuos pero no sezga el movimeinto de los agentes según esta jerarquía, en contraste, por ejemplo, al algoritmo GWO [32][33], no existe un individuo que influencie en mayor grado el comportamiento del cardumen. La mejor solución del algoritmo es la posición de la presa más grande. La cantidad de comida disponible por presa disminuye por el número histórico de presas que se han

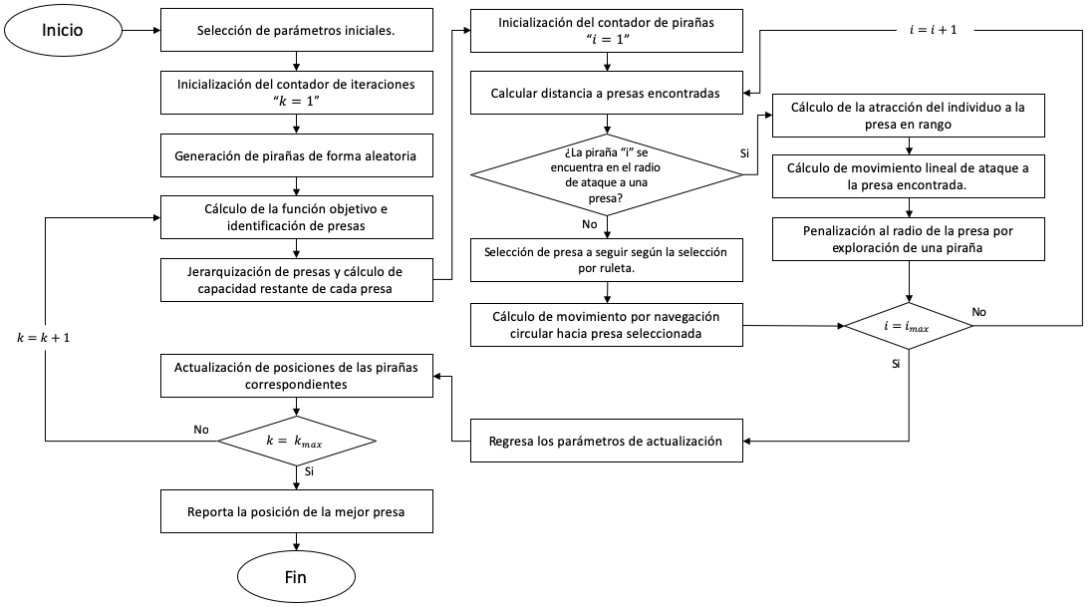

FIGURA 3.3: Diagrama de Flujo del algoritmo PFO

alimentado de ella, es decir que la han atacado; este recurso es finito. Dicha cantidad solamente disminuye, es decir que es imposible que aumente entre iteraciones. Esto es similar al comportamiento del modelo de enfriamiento del algoritmo de recocido simulado [59]. Esta característica es implementada para incentivar la exploración de pirañas en otras áreas una vez que la vecindad de la localización de la presa fue explorada, de esta forma se incentiva la exploración cercana y si no se encuentran nuevas y mejores presas, se incentiva la salida de las pirañas a otras áreas.

Este recurso de comida disponible por presa determina qué tan atractiva resulta para cada piraña a través de la reducción de la vecindad que provoca un ataque. Entre menor sea la cantidad de comida disponible en una presa, más pequeño será el radio permitido para su ataque al reducir su vecindad de atracción.

El flujo del algoritmo se observa en la figura 3.3; se ofrece una explicación sintetizada del mismo a continuación.

El diagrama en la figura 3.3 se explica a continuación:

1. Los sujetos se inicializan en una posición aleatoria en el espacio de búsqueda.
2. Cada sujeto es evaluado en la función objetivo, las posiciones que arrojen un mejor resultado se guardan como las j presas iniciales.
3. Se inicializa un parámetro de capacidad de la presa, es decir que cada presa tendrá una capacidad máxima de alimento que ofrecer.
4. Las pirañas, se mueven hacia las presas; esto sucede mediante tres factores principales, social, individual y aleatorio. Esta es la etapa normal de navegación.

4.1 Factor social: El factor social será la señal emitida desde la posición de la presa, la intensidad de la señal es inversamente proporcional a la distancia de la piraña a cada presa, la dirección a la que se dirigirá es determinada por una probabilidad de selección proporcional a la cercanía de la piraña a la presa.

4.2 Factor individual: El factor individual se calcula de acuerdo al "alimento" restante de cada presa. El radio de atracción de cada presa es recalculado simulando los recursos restantes en ella; radio que disminuye progresivamente al ser multiplicado por el hiperparámetro de castigo a la exploración z.

4.3 Factor aleatorio: Para evitar mínimos o máximos locales, el factor aleatorio se introduce como una probabilidad de selección por ruleta. De forma similar a la selección de ruleta en un algoritmo genético para la selección de soluciones [20][50][11], se selecciona en que dirección mover a una piraña en el factor social, no solamente por su cercanía a cada presa, sino por un factor aleatorio que influye sobre cual presa deberá perseguir, incentivando la exploración aleatoria del espacio.

5. Ataque: Si una piraña se encuentra en el radio calculado en el punto 4.2, le será permitido realizar un ataque a la presa. Esta condición sobre escribe la navegación normal y utiliza una política de movimiento linealmente acelerado hacia la presa en cuestión.

6. Actualización de posiciones: Con los factores anteriores; se actualiza la posición de cada individuo y se reinicia el flujo hasta terminar las iteraciones permitidas.

3.1.3. Creación del cardumen

En el algoritmo de PFO, cada piraña tiene una localización en el espacio de la función de búsqueda; esta posición es una solución de i dimensiones en el espacio de soluciones. Es posible entonces expresar el conjunto de pirañas como en la matriz (3.2):

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & \cdots & p_{1,i} \\ p_{2,1} & p_{2,2} & \cdots & \cdots & p_{2,i} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{n,1} & p_{n,2} & \cdots & \cdots & p_{n,i} \end{bmatrix} \tag{3.2}$$

En el que n es el número de pirañas que integrarán el cardumen; i es la cantidad de dimensiones que forman el espacio de soluciones. Los valores de evaluación de la solución en la función de cada piraña son guardados en el vector Pf, mostrado a continuación en (3.3):

$$f(P) = Pf = \begin{bmatrix} Pf_1 \\ Pf_2 \\ . \\ . \\ Pf_n \end{bmatrix} \tag{3.3}$$

El siguiente componente, la matriz compuesta por las posiciones de las presas encontradas (3.4), será:

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & ... & ... & r_{1,i} \\ r_{2,1} & r_{2,2} & ... & ... & r_{2,i} \\ . & . & . & . & . \\ . & . & . & . & . \\ r_{j,1} & r_{j,2} & ... & ... & r_{j,i} \end{bmatrix} \tag{3.4}$$

Donde j es la cantidad de presas permitidas por los parámetros iniciales del algoritmo. Por último, el ajuste o evaluación de cada solución en la matriz R se encuentra en el vector Rf:

$$f(R) = Rf = \begin{bmatrix} Rf_1 \\ Rf_2 \\ . \\ . \\ Rf_j \end{bmatrix} \tag{3.5}$$

3.1.4. Inicialización y actualización de posiciones de las pirañas

La inicialización de las pirañas en el espacio de soluciones se da de forma aleatoria, acotando los valores posibles en cada dimensión según un límite superior e inferior proporcionados por el rango de la función explorada. Esta inicialización puede ser expresada en la ecuación (3.6) de la forma:

$$P(i,j) = (Ub(i) - Lb(j)) \times rand(0,1) + Lb(i) \tag{3.6}$$

Donde Ub y Lb son vectores con los límites superiores e inferiores en cada dimensión del espacio de soluciones; $rand(0,1)$ será un número aleatorio entre 0 y 1.

Para la actualización de la posición de cada piraña, se contemplan dos factores discutidos anteriormente. Como se muestra en el diagrama de flujo de la figura 3.3, luego de la inicialización, la evaluación de las soluciones en la función explorada y la identificación de las presas (mejores soluciones) sigue el cálculo de distancias entre pirañas y presas, que se muestra a continuación. Se calcula la distancia euclidiana entre la piraña correspondiente y cada presa, esta matriz de distancias será la matriz (3.7):

$$D_{pr} = \begin{bmatrix} d_{1,1} & d_{1,2} & ... & ... & d_{1,j} \\ d_{2,1} & d_{2,2} & ... & ... & d_{2,j} \\ . & . & . & . & . \\ . & . & . & . & . \\ d_{n,1} & d_{j,2} & ... & ... & d_{n,j} \end{bmatrix} \tag{3.7}$$

Donde n es la cantidad de pirañas en el cardumen y j es el número de presas definidas. Cada $d_{n,j}$ representa la distancia euclidiana calculada entre la piraña n y la presa j.

Posteriormente se compara cada distancia para identificar si alguna de ellas es menor al radio de atracción de cada presa. A cada presa se le asigna un radio máximo llamado w según su posición en la lista de j presas permitidas. Este radio es asignado según la cantidad de presas y su posición en la lista. De forma que primero se calcula la diferencia

que habrá entre el radio de cada presa según la expresión (3.8):

$$\Delta\, w = \frac{1}{j} \tag{3.8}$$

Posteriormente cada radio esta calculado de acuerdo a la ecuación (3.9), mostrada a continuación. En la cual k es la posición de la presa en el vector ordenado (de forma ascendente o descendente, según la tarea de optimización aplicada) de acuerdo a los resultados de la evaluación en la función de búsqueda en Rf; entonces k para la primera presa en el vector será igual a 1, para la siguiente presa será igual a 2 e incrementando sucesivamente.

$$w_k = 1 - (\Delta\, w \times (k-1)) \tag{3.9}$$

De ser el caso, significa que la piraña se encuentra dentro del radio de ataque a la presa, este es el criterio que decidirá el tipo de movimiento que seguirá en la actual iteración este agente.

3.1.5. Fase de ataque

En este caso de movimiento, se modela un acercamiento con movimiento lineal hacia la presa, inspirado en el método de ataque individual de las pirañas [15] [31]. Este movimiento de desplazamiento lineal esta dado por la velocidad de desplazamiento en cada dimensión, expresada por la ecuación (3.10):

$$V_{n,i} = V_{n,i} + rand(0,1) \times b \times (R_{j,i} - P_{n,i}) \tag{3.10}$$

En la que $V_{n,i}$ es la velocidad de movimiento de la piraña en la dimension i, $rand(0,1)$ es un número aleatorio entre $(0,1)$, b es el factor constante de velocidad de ataque mostrado en el Cuadro 3.2, $R_{j,i}$ es la posición de la presa atacada en la dimensión i y $P_{n,i}$ es la posición actual de la piraña atacante en i.

La posición actualizada de la piraña será la posición anterior más la nueva velocidad calculada previamente en (3.10) y se muestra en la ecuación (3.11):

$$P_{n,i} = P_{n,i} + V_{n,i} \tag{3.11}$$

Por último, se realiza la reducción de los radios de las presas de acuerdo la ecuación (3.12):

$$w_k = z \times w_k \tag{3.12}$$

Donde z es el factor de castigo a la sobre explotación presentado en la tabla 3.2 y w_k es el radio previamente calculado en las ecuaciones (3.8) y (3.9).

3.1.6. Fase de navegación

Las pirañas que no se encuentren en el radio de atracción de una presa, se dice que se encuentran en fase de navegación. Durante esta fase, se calcula una probabilidad de que cada piraña se mueva hacia una presa encontrada, esta probabilidad se utilizará para la selección por ruleta mostrada en (3.13):

$$Prob(n,j) = 1 - \frac{d(n,j)}{\sum_{j=1}^{j=j_{max}} d(n,j)} \tag{3.13}$$

Tal que la probabilidad de selección será inversamente proporcional a la distancia entre piraña y presa. Además, esto garantiza que la suma de las distancias a cada presa de una misma piraña será 1.

Entonces es posible construir una matriz cuyos elementos son probabilidades calculadas en (3.14):

$$MProb_{pr} = \begin{bmatrix} prob_{1,1} & prob_{1,2} & ... & ... & prob_{1,j} \\ prob_{2,1} & prob_{2,2} & ... & ... & prob_{2,j} \\ . & . & . & . & . \\ . & . & . & . & . \\ prob_{n,1} & prob_{j,2} & ... & ... & prob_{n,j} \end{bmatrix} \tag{3.14}$$

Para la cual para cada n en $MProb$ se arrojará un número aleatorio entre $(0,1)$ que será comparado contra los elementos de la matriz de probabilidad. La presa seleccionada será aquella en la que se cumpla la condición (3.15):

$$rand(0,1) >= MProb(n,j) \tag{3.15}$$

Resultando en un vector (3.16) que contiene la presa seleccionada a seguir para cada piraña:

$$SR = \begin{bmatrix} sr_1 \\ sr_2 \\ . \\ . \\ sr_n \end{bmatrix} \tag{3.16}$$

Una vez identificada la presa a la que se dirigirá cada piraña, su movimiento hacia ella esta modelado utilizando un espiral logarítmico sujeto por las siguientes condiciones:

- El punto inicial del espiral será la posición de la piraña.
- El punto final del espiral será el la posición de la presa seleccionada.
- Los cambios en el rango del espiral no deben sobrepasar el espacio de búsqueda.

El movimiento es descrito según la ecuación (3.17):

$$M(n,j) = d_{n,j} \times e^{ct} \times cos(2\pi t) + R_j \tag{3.17}$$

TABLA 3.2: Tabla de parámetros ajustables en PFO

Nombre del Parámetro	Rango	Explicación e impacto
n	$n > 0$	Número total de pirañas en el cardumen
j	$0 < j < n$	Numero máximo de presas permitidas para el cardumen
b	$b > 0$	Factor de velocidad en ataque
c	$c > 0$	Parámetro que define la velocidad de convergencia del espiral en la fase de navegación, modifica que tan rápido disminuye el radio del espiral
z	$0 < z < 1$	Factor de castigo a la exploración de una presa, degradación del radio de ataque

Donde c es un parámetro de entrada mayor a 0, descrito en la tabla 3.2, que define la forma del espiral logarítmico; t es un valor aleatorio entre $(-1, 1)$ y $d_{n,j}$ la distancia entre la piraña n y la presa j. R_j es la posición de la presa j a la que se dirige la piraña n.

Todos los parámetros ajustables o hiperparámetros utilizados en el algoritmo se listan y explican en la tabla 3.2. Adicionalmente, en la tabla 4.8 Se muestra una comparación de las características de diseño del algoritmo presentado, contrastándolo con los algoritmos metaheurísticos de referencia.

3.2. Pseudocódigo de la propuesta

Haciendo referencia a las figuras 3.3, a continuación se presenta una secuencia de pseudocódigo de la propuesta. La explicación detallada del algoritmo PFO ya fue presentada anteriormente, por lo que en esta sección del documento, no se discutirán los detalles del algoritmo. Los pasos previos al entrenamiento, como la codificación de variables y la aplicación de técnicas de tratamiento de datos no son parte del algoritmo; por lo que no se incluyen en el pseudocódigo. En el algoritmo descrito, P es la población total de n pirañas, R es el conjunto de j presas permitidas, b es el factor de escalamiento de velocidad de ataque, c el factor de velocidad de convergencia del espiral para navegación, w es el radio de ataque para cada presa y z es el factor de penalización al radio de la presa; la función f es la evaluación de la función objetivo, D es la función de distancia entre pirañas y presas, A es la función de actualización de movimiento de ataque, M es la función de actualización de movimiento de navegación.

3.3. Paralelización

La ejecución en paralelo de algoritmos ha sido explorada en distintos marcos de trabajo. Entre las que destaca la parelilzación de la ejecución en múltiples hilos en un solo procesador (CPU), la paralelización masiva en CUDA dentro de una GPU [47] y la distribución de tareas entre ejecutores utilizando herramientas como Spark [16] [49].

Respecto a la paralelización y distribución, CUDA delega operaciones a distintos núcleos en una GPU, implementando controles de serialización que permiten que la ejecución se haga de forma ordenada entre todos los núcleos [47] [23].

En contraste, Spark trabaja enfocándose a la distribución de cómputo, esto quiere decir que, asigna trabajos a una máquina virtual ejecutora completa [16] [49]. Esta máquina

Algorithm 1 Piranha Feast Optimization, PFO

Require: $n > 0$
Require: $0 < j < n$
Require: $b > 0$
Require: $c > 0$
Require: $0 < z < 1$
Require: $k_{max} > 1$
  $\overline{P} = [p_1, p_2, ..., p_n], p_n \in \mathbb{R}^i$
  $k = 1$
  while $k < k_{max}$ **do**
    $\overline{R} = f(\overline{P}) = [f(p_1), f(p_2), ..., f(p_n)] = [r_1, r_2, ..., r_n]$
    Ordenar $\overline{R} \leftarrow [r_{max}, ..., r_{min}]$
    $\overline{W} \leftarrow$ Primeros j elementos de los argumentos de $\overline{R}$
    $\overline{W} = [r_{max}, ..., r_j]$
    $h = 1$
    while $h < n$ **do**
      $d_h = D(p_h, \overline{W})$
      if $d_h < w_j$ **then**
        $a_h = A(p_h, r_j)$
        $p_h = p_h + a_h$
        $w_j = w_j \times z$
      else
        $r_j \leftarrow$ Selección por ruleta de una presa en $\overline{W}$
        $n_h = N(p_h, r_j)$
        $p_h = p_h + n_h$
      end if
      $h = h + 1$
    end while
    $k = k + 1$
  end while
  return Primer (mejor) elemento de $\overline{W}$

contiene sus propios recursos asignados, todo es coordinado por una máquina supervisora, así aunque un ejecutor termine el trabajo antes que otras, no se presentan problemas de continuidad. Se debe considerar la capacidad de memoria, almacenamiento y núcleos de los ejecutores ya que cada uno recibe una parte de la base de datos, y la aceleración de procesamiento dependerá de los recursos alojados a cada ejecutor.

En una arquitectura distribuida [30], cada nodo ejecutor recibe una partición de la base de datos sobre la cual realizará cálculos. Luego de esto, un nodo maestro u orquestador entrega una cola de operaciones de forma serializada a cada ejecutor y este se dedicará exclusivamente a procesar esa cola de operaciones sobre sus datos disponibles. Una vez que se termina la cola en cada ejecutor, es posible recolectar los resultados desde el nodo maestro que los mantendrá accesibles [30] [16]. En contraste, una estrategia de paralelización en múltiples hilos debe crear una cola de operaciones, luego identificar qué operaciones son interdependientes y cuáles pueden ser realizadas sin necesidad de un resultado anterior [61]. Una vez construidas estas colas, cada una es entregada a un hilo de procesamiento, que realizará las operaciones y guardará los resultados en memoria para que una vez terminadas todas las subdivisiones de las colas originales, los resultados sean accesibles.

La estrategia de paralelización presenta una ventaja para el desarrollo de nuevos modelos y algoritmos sobre la estrategia distribuida; la disponibilidad de la información para todo el sistema. Dada la forma de operar de los marcos distribuidos, cada ejecutor trabaja sobre una partición del conjunto de entrenamiento [30], dificultando la tarea de aprendizaje ya que cada ejecutor comienza entrenamientos independientes según su partición. En contraste, una paralelización local reparte colas de tareas y no particiones de datos; lo que la vuelve ideal para desplegar nuevos algoritmos o modificados.

En la paralelización implementada se ha decidido utilizar paralelización por hilos en el procesador local. Esto debido a que la migración de código serializado ha paralelizado en estos recursos resulta sencillo utilizando Numba; además de proporcionar la primera línea comparativa de desempeño contra el desarrollo futuro utilizando CUDA.

3.4. Codificación del algoritmo

La propuesta contempla implementar la paralelización del algoritmo, integrando esta arquitectura a la actual estrategia, minimizando el impacto del aumento de cálculos ocasionado por las iteraciones de los algoritmos metaheurísticos de población. Para esta implementación del algoritmo será utilizado el marco de paralelización en procesador a través de su interfaz en Python, llamada Numba. La razón de la selección de tecnología es la existencia de amplia documentación de Numba para paralelización en procesadores, la popularidad y conveniencia de dicho lenguaje de programación, el acceso a hardware compatible con estas tecnologías y a software de depuración especializado, así como la posibilidad de expandir el alcance de la paralelización en el futuro con la implementación de paralelización en una unidad de procesamiento gráfico de forma nativa usando Numba.

El código generado en este trabajo se encuentra disponible en un repositorio público en "GitHub". Este repositorio incluye todos los algoritmos metaheurísticos en su modalidad paralelizada y serializada, el código de entrenamiento para el caso de prueba de reconocimiento de actividades humanas y el generador de funciones unidimensionales y bidimensionales utilizadas. Adicionalmente, se incluye la implementación del algoritmo de AHN que fue utilizado. El repositorio se encuentra en la dirección: https://github.com/jggmemo/pfo_ahn

3.5. Métricas y soporte de la propuesta

Existen algoritmos metaheurísticos actuales que podrían ser utilizados para el mismo efecto que el discutido, por lo que las pruebas contemplan la integración de dichas alternativas, previo al diseño del algoritmo propuesto. De esta forma se evaluarán las ventajas y desventajas de cada algoritmo, y la propuesta será configurada respecto a estos resultados para su mejoramiento. Se realizarán pruebas con tres algoritmos, GWO, PSO y BA.

La propuesta será soportada por estudios de tiempo y precisión para la implementación original del modelo y para la implementación modificada. Dichas pruebas de rendimiento se realizaron con los siguientes datos de entrada:

a) Una función sintética de prueba de una sola dimensión con 1000, 10000 y 100000 registros producto de un proceso de muestreo aleatorio sobre la función original (3.18), mostrada en la figura 3.4, donde x es la variable de entrada al modelo y f(x) su salida.

$$f(x) = \begin{cases} \arctan(\pi x) & -1.0 \leq x < 0.1 \\ \sin(\pi x) & 0.1 \leq x < 0.6 \\ \cos(\pi x) & 0.6 \leq x \leq 1.0 \end{cases} \tag{3.18}$$

b) Una función sinodal bidimensional con 1000, 10000 y 100000 registros producto de un proceso de muestreo aleatorio sobre la función original (3.19), mostrada en la figura 3.5, donde x_1 y x_2 son las variables de entrada al modelo y y_1, o $f(x_1, x_2)$, es el resultado.

$$F : \begin{cases} x_1 = \cos(t) \\ x_2 = t \\ y_1 = \sin(t) \end{cases} , t \in [0, 20] \tag{3.19}$$

Todos los experimentos realizados con las funciones muestreadas se realizaron reservando 70 % de las observaciones para entrenamiento y el 30 % restante para pruebas del modelo entrenado. Además de realizar 10 validaciones durante el proceso de entrenamiento.

El caso de estudio se realizará sobre la base de datos "UP-Fall Detection" [29], que será una prueba de clasificación. El conjunto de datos incluye 11 actividades, 5 tipos de caídas. Los datos de la base son lecturas de 14 sensores, 6 infrarrojos, 2 cámaras, 1 EEG headset y 5 sensores vestibles. Al final, contiene 32,294 registros con 756 características o dimensiones y 12 categorías meta.

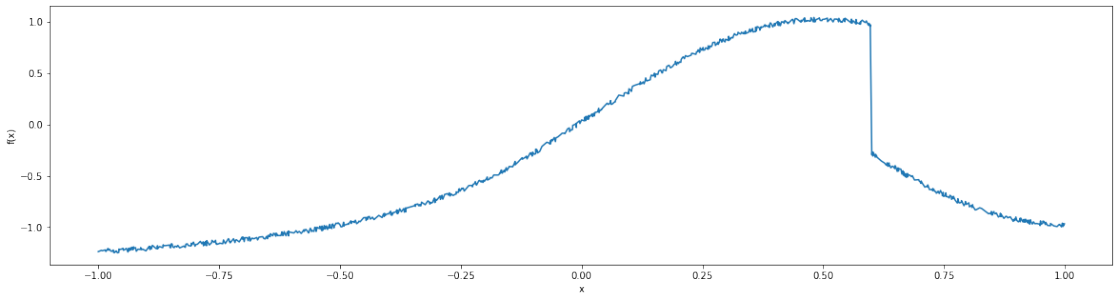

FIGURA 3.4: Función unidimensional utilizada. Donde x es la variable de entrada.

Para la evaluación de las funciones de referencia se mide el tiempo de ejecución en la fase de entrenamiento y el error cuadrático medio, o RMSE por sus siglas en inglés, de la regresión resultante en cada experimento. La ecuación para calcular el RMSE se presenta en (3.20).

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{3.20}$$

Donde $\hat{y}_i$ son los valores resultantes de la predicción, y_i son los valores reales de las observaciones y n es el número total de observaciones.

Con el fin de validar el impacto de la estrategia de paralelización, por cada 10 ciclos de entrenamiento para cada configuración se mide el RMSE promedio, la desviación estándar del RMSE, tiempo de ejecución promedio y desviación estándar del tiempo de ejecución.

Para la prueba de clasificación en el caso de estudio en HAR, se evaluarán las siguientes métricas:

1. Exactitud (accuracy): [54] La exactitud general del modelo, en un ejercicio de clasificación multiclase, se reporta como el promedio de la exactitud específica en cada clase. Calculada simplemente como la división de las observaciones correctamente clasificadas debajo de cierta categoría entre el total de observaciones reales por clase. Es simplemente un agregado que muestra la cantidad de observaciones clasificadas que hay en cada clase según su total particular. Se define de la forma:

$$Acc_c = \frac{1}{N_s} \sum_{i=0}^{N_s-1} 1(Yp_i = Ya_i) \tag{3.21}$$

   Donde Acc_c es la exactitud de cada clase, N_s es el número total de observaciones por clase, Yp_i es el elemento i del vector de predicciones y Ya_i el elemento i del vector de categorias reales. La métrica final en el ejercicio multi-clase es simplemente el promedio de todos los valores resultantes en el conjunto $\overline{Acc}$.

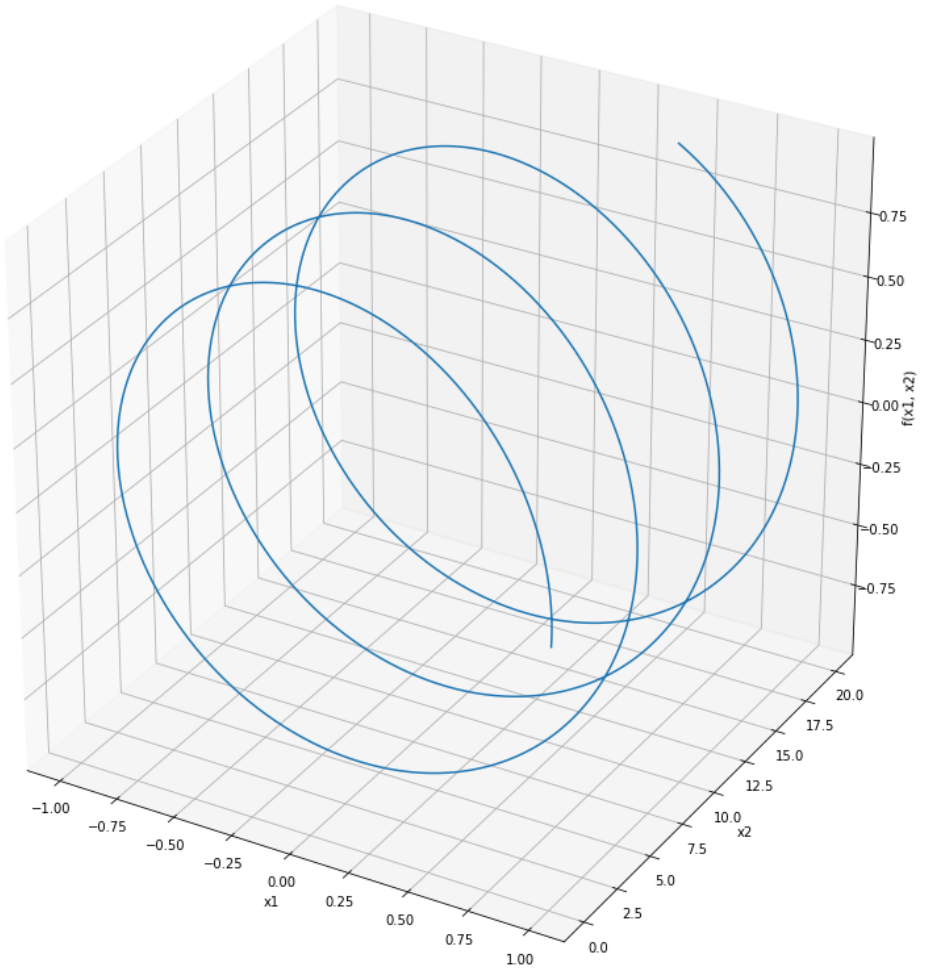

FIGURA 3.5: Función bidimensional utilizada. Donde x_1 y x_2 son las variables de entrada.

2. Precisión macro (precision): [54] En un problema de clasificación multiclase la precisión se calcula por clase; indicando cual es la proporción de clasificaciones correctas (verdaderos positivos) entre el total de observaciones clasificadas como pertenecientes a cada clase sin importar si son verdaderos positivos o falsos positivos; por lo que esta métrica ayuda a evaluar la capacidad del modelo para predecir la pertenencia de una observación a la predicción otorgada, en otras palabras, responde a ¿qué porcentaje de las predicciones hechas fueron correctas? Se define:

$$Pr_c = \frac{Tp}{Tp + Fp} \tag{3.22}$$

Donde Pr es la precisión resultante por clase, Tp es la cantidad de verdaderos positivos o predicciones correctas de pertenencia a una clase en particular, Fp es

el conjunto de falsos positivos o predicciones incorrectas de pertenencia a una clase en particular. La métrica final, de tipo macro, en el ejercicio multi-clase es simplemente el promedio de todos los valores resultantes en el conjunto $\overline{Pr}$.

3. La puntuación F1 macro o valor F macro (macro F1 score): Por sí sola, la precisión no es suficiente para evaluar un modelo de clasificación, en especial cuando se trata de un modelo de aplicación médica. La razón es que la precisión no mide la proporción de falsos negativos en un experimento. Entonces, es necesaria una métrica que se vea afectada por la proporción de observaciones erróneamente catalogadas como no pertenecientes a su clase real. Existe una métrica complementaria que mide exactamente la proporción de falsos negativos, la exhaustividad, en ingles llamada *recall*. Pero al igual que sucede con la precisión, por si misma no es suficiente para evaluar el modelo. Por esta razón fue seleccionada la puntuación F1 como tercera métrica. La puntuación F1 combina la precisión y la exhaustividad del modelo en un único número que permite evaluar el desempeño de un modelo y alcanzar un balance entre ambas componentes [54]. La puntuación F1 se calcula como la media armónica de las dos componentes, de la forma:

$$F1_c = \frac{2 \times Pr \times Rc}{Pr + Rc} \tag{3.23}$$

   Donde Pr y Rc son respectivamente la precisión y exhaustividad del modelo. mientras que $F1_c$ es la puntuación F1 por categoría meta. La métrica $F1$ final es el valor promedio de los elementos en el vector $\overline{F1_c}$

4. Tiempo de entrenamiento: Es simplemente el tiempo medido en segundos desde el momento en el que comienza la primera iteración de entrenamiento hasta que concluye la última iteración.

El término "macro" se refiere a que la métrica es calculada tomando el valor promedio del conjunto de exactitudes por clase, dándole el mismo peso o importancia a cada una, sin importar su representación; en contraste, las métricas micro buscan dar un peso igual a cada instancia u observación, lo que provoca que las clases con mayor representación tengan mayor peso. En este caso no es deseable usar la métrica "micro" ya que nuestra base de datos esta desbalanceada. Lo anterior será explorado más a fondo en capítulos siguientes. La razón principal para la selección de las métricas anteriores es la relevancia de ellas para la naturaleza de la aplicación. Al ser una aplicación médica y de salud, es necesario penalizar las clasificaciones falsas, específicamente en casos falsos negativos de caídas.

3.6. Diseño de experimentos

Se realizarán 4 secciones de experimentación:

1. Referencia: Se realizan pruebas del algoritmo serializado en la función unidimensional y bidimensional propuestas. Se entrena el modelo con 3, 10 y 20 moléculas, 1000, 10000 y 100000 registros, con 100 validaciones por entrenamiento y se prueban 3 algoritmos de optimización metaheurística y el algoritmo de gradiente; se

entrenará 10 veces cada configuración para recoger datos de promedio y desviación estándar del tiempo de entrenamiento y error cuadrático medio.

2. Paralelización: Se realizan pruebas del algoritmo paralelizado en la función unidimensional y bidimensional propuestas. Se entrena el modelo con 3, 10 y 20 moléculas, 1000, 10000 y 100000 registros, con 100 validaciones por entrenamiento y se prueban 3 algoritmos de optimización; se entrenará 10 veces cada configuración para recoger datos de promedio y desviación estándar del tiempo de entrenamiento y error cuadrático medio.
3. Una vez desarrollado e implementado el algoritmo de PFO, se repetirá la rutina de pruebas previamente descrita de forma serializada y posteriormente paralelizada.
4. Caso de estudio: Se realizan cinco nuevas pruebas sobre la base de datos "UP-Fall Detection Dataset" [29].

Esto constituye un total de 1625 experimentos realizados en aproximadamente 80 horas de procesamiento total.

3.7. Implementación y ambiente de pruebas

3.7.1. Hardware:

Todas las pruebas se realizaron en una misma computadora con las siguientes características:

Procesador: Intel Core i7-7820HK @ 4.3 GHz

Memoria RAM: 16384 MB, 2x 8 GB DDR4-2400, Dual-Channel, 2/4 puertos en uso.

Almacenamiento: Toshiba NVMe THNSN5512GPU7, 512 GB, SSD + HGST Travelstar 7K1000 HTS721010A9E630, 1TB HDD @ 7200 rpm.

3.7.2. Software:

El desarrollo de los algoritmos, la programación de la secuencia de pruebas y recolección de resultados se realizó en Anaconda 2018.12 con Python 3.7. Las librerías utilizadas específicamente y sus versiones se enlistan a continuación:

1. Numpy - 1.16.2 [39]: Permite la manipulación de matrices y es compatible con la compilación JIT de Numba.
2. Niapy - 1.0.2 [60]: Librería de algoritmos metaheurísticos inspirados en la naturaleza. Se utilizaron como base para las pruebas serializadas y fue modificada para habilitar su ejecución en paralelo compatible con Numba.
3. Numba - 0.43.0 [25]: Librería capaz de generar código máquina interpretando notación de Python. Habilita ejecución en paralelo de instrucciones iterativas y recursivas sobre objetos de numpy.

Capítulo 4

Resultados y discusión

En el presente capítulo se discuten, a través de las gráficas y tablas presentadas, los resultados obtenidos. Inicialmente se discuten los resultados de tiempo de ejecución. Posteriormente se realiza la comparación del error de cada algoritmo; en primer lugar los resultados serializados seguidos de los paralelizados. Cerrando este capítulo se exploran los resultados de la precisión en cada conjunto experimental y la aplicación del desarrollo realizado al problema HAR.

Durante todo el capítulo se encuentran tablas de resultados con las siguientes columnas:

- ET_Mean: Es el tiempo de ejecución promedio de las observaciones experimentales realizadas. Medido desde el momento en el que el modelo comienza su entrenamiento hasta que se realiza la última iteración.
- ET_stDev: Es la desviación estándar del conjunto de medidas de tiempo de ejecución.
- Moléculas: La cantidad de moléculas construidas en el modelo de AHN.
- RMSE_Mean: Es el promedio del error cuadrático medio de las observaciones experimentales realizadas.
- RMSE_stDev: La desviación estándar de las mediciones del error cuadrático medio.
- Reg_Number: Cantidad de registros que conforman la función sobre la cual se realiza la regresión; la función que el modelo aprende a predecir.

Durante todos los experimentos se mantuvo la misma configuración de hiperparámetros para cada algoritmo metaheurístico. Lo anterior fue decidido para poder realizar comparaciones entre experimentos y, aunque es posible alcanzar mejores resultados cambiando algunos parámetros como el total de partículas, factores de crecimiento internos y máximos y mínimos de velocidades en cada caso; incluir estos cambios por caso haría que no fuera posible comparar métricas como el tiempo de ejecución ya que también cambiarían los recursos computacionales requeridos.

El algoritmo de PFO se configuró con 150 pirañas activas, 3 presas permitidas, un factor de velocidad de ataque de 1.25, la velocidad de convergencia fue de 0.75 y el factor de penalización al radio de las presas fue de 0.9. Estos parámetros fueron decididos basados en la experiencia de uso del algoritmo.

El algoritmo BA fue configurado de acuerdo a los resultados presentados por Xue [63], con pequeñas variaciones para ajustarse de mejor forma a la función utilizada. El rango de frecuencia se estableció con un mínimo de 0 y un máximo de 15. El volumen A seleccionado fue de 1,0 y la velocidad a la que es emitido un pulso r fue de 0.8. Los parámetros a y g fueron seleccionados como 0,99 y 0,9 respectivamente.

El algoritmo GWO fue configurado con los parámetros originales presentados en [33] y en las aplicaciones documentadas en [14]. La población total de lobos, que es en realidad el único parámetro controlable en al implementación utilizada del algoritmo fue de 50 lobos.

Por último, el algoritmo PSO cuenta con extensa investigación sobre sus parámetros y configuraciones. Usando los resultados de Thomas Beielstein en [2], donde explora la configuración de los parámetros de PSO. La población seleccionada es de 100 partículas, el rango de velocidad permitida se configuro entre 0,1 y 0,8, el peso de la inercia w se configuro a 0,9. El componente cognitivo "C1" fue de 0,75 mientras que el componente social "C2" seleccionado fue de 2,25.

Adicionalmente, algunas métricas en las tablas de resultados se encuentran con las siglas "NA", correspondientes a "No Aplicable". Esto se debe a que son métricas que dependen de la comparación entre dos o más observaciones; y en ciertos casos no fue posible reproducir el experimento de forma exitosa en más de una ocasión debido a la inestabilidad del entrenamiento.

4.1. Análisis de impacto en el tiempo de ejecución

4.1.1. Experimentación serializada

Respecto al tiempo de ejecución, es importante recalcar la relación con el incremento de tiempo según los siguientes factores:

1. Mayor número de moléculas en la arquitectura del modelo
2. Mayor número de puntos en los vectores de entrada y salida
3. Mayor número de dimensiones o variables de entrada

La primera observación respecto al número de moléculas se presenta al comparar el tiempo de ejecución de cada algoritmo respecto a sí mismo, con el mismo número de registros del vector de entrada y la misma dimensionalidad. Esto quiere decir que al comparar el tiempo entre el entrenamiento de un modelo de 3 moléculas contra un modelo de 20 moléculas, es claro que el tiempo de entrenamiento aumentará. Este incremento puede llegar a ser de hasta 3.6 veces en el caso del entrenamiento con el algoritmo basado en el gradiente, mostrado en las tablas A.1 y A.2.

La segunda observación es clara al concentrarnos en el aumento de tiempo presente entre un mismo algoritmo con mismo número de moléculas en su arquitectura pero distinto número de registros. El caso de mayor crecimiento es el incremento de aproximadamente 14 veces entre el modelo de 3 moléculas entrenado con el algoritmo del gradiente, comparado al entrenarlo con 10,000 registros y con 100,000 registros. Aumento que se presentó en los casos de una y dos dimensiones de las tablas A.1 y A.2.

Adicionalmente, al evaluar los resultados del entrenamiento de una función unidimensional contra una función bidimensional encontramos la tercera observación importante. En todos los casos, incluso en los entrenamientos con algoritmos metaheurísticos, es aparente que el aumento dimensional también incrementa el tiempo total. Estas tres observaciones son importantes ya que se mantuvieron entre todos los resultados encontrados, incluso ante los resultados paralelizados.

También se encontró que el tiempo de ejecución más largo se presentó con el algoritmo basado en el gradiente, entrenando un modelo de 20 moléculas con 100,000 registros. Tal como se esperaba, este algoritmo aumenta el tiempo de entrenamiento dramáticamente frente a los acercamientos de algoritmos metaheurísticos. Adicionalmente se encontraron múltiples problemas de consistencia, por lo que no fue posible replicar esta instancia del experimento múltiples ocasiones. La reducción del tiempo entre el algoritmo del gradiente y un algoritmo metaheurístico se observó en mayor grado durante el entrenamiento de 20 moléculas con 100,000 registros en dos dimensiones de la tabla A.2 con el algoritmo BA en dos dimensiones de la tabla A.4; donde se logró una reducción de 266 % aproximadamente del tiempo. Son claros entonces los beneficios en términos de tiempo de entrenamiento que presentan los algoritmos metaheurísticos frente a los algoritmos basados en el gradiente.

Los resultados presentados en las tablas A.9 y A.10 muestran un patrón interesante en el tiempo de ejecución del algoritmo PFO, en el que consistentemente el algoritmo es más lento que los demás en los casos más pequeños, con menos moléculas que entrenar. Progresivamente esta diferencia va disminuyendo, hasta que en los casos más complejos, con más moléculas y registros, el algoritmo PFO muestra ser más rápido que todos los demás algoritmos probados en ese mismo caso de prueba en las tablas A.4, A.6 y A.8. Esto sugiere que el algoritmo PFO es capaz de escalar de mejor forma que los otros métodos comparados, ya que su tiempo de ejecución incrementa más lento que los otros algoritmos en las mismas pruebas. Además, los resultados son consistentes dada una baja desviación observada entre los distintos experimentos. El fenómeno anterior también es observado en las tablas de resultados de una sola dimensión, tablas A.3, A.5 y A.7, en las que, aunque con una diferencia más pequeña, el incremento en tiempo entre cada caso de prueba es mayor en los algoritmos BA, PSO y GWO respecto al incremento observado en PFO.

Ahora realizaremos una comparación entre los resultados de los algoritmos metaheurísticos en las tablas. De este conjunto, el experimento más lento fue el entrenamiento de un modelo de 20 moléculas con 100,000 registros utilizando el algoritmo PSO para una función bidimensional. Donde se encontró que el tiempo aumentó casi en 40 % entre el experimento de una dimensión y de dos dimensiones. En la tabla 4.1, fueron seleccionados los conjuntos experimentales más complejos para cada algoritmo; donde el caso más complejo es el conjunto de mayor número de registros en la función bidimensional y con el modelo AHN formado por 20 moléculas. Estos resultados son graficados en la figura 4.1; de la cual se observa una reducción aproximada del 50 % en el tiempo de entrenamiento respecto al algoritmo del gradiente y en el caso de los algoritmos GWO y PSO, un aumento sustancial en la consistencia de ejecución y resultados. El algoritmo PFO se posiciona como el segundo método más rápido, justo debajo del algoritmo BA, mostrando nuevamente su capacidad de ser escalado a problemas más demandantes.

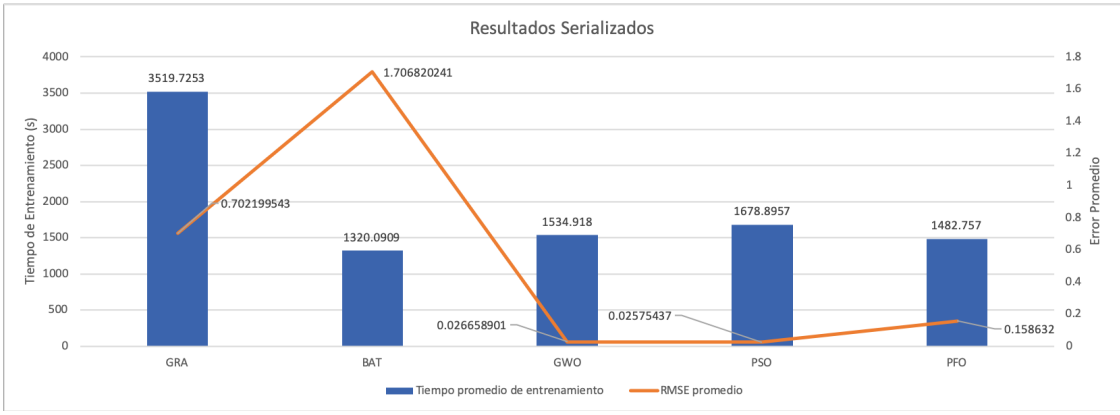

FIGURA 4.1: Gráfica comparativa de resultados en el conjunto experimental más complejo de la experimentación serializada y una función bidimensional

TABLA 4.1: tabla comparativa de resultados serializados. Muestra las mediciones recogidas en la experimentación bidimensional y con la mayor cantidad de moléculas y registros. Es el conjunto experimental más exigente.

Comparativo de resultados serializados					
Reg_Number	**Algoritmo**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
100000	GRA	3519.7253	NA	0.702199543	NA
	BA	1320.0909	8.650415372	1.706820241	0.225193069
	GWO	1534.918	9.869435792	0.026658901	0.010769453
	PSO	1678.8957	7.9837484	0.02575437	0.009292395
	PFO	1482.757	8.466973	0.158632	0.0155679

4.1.2. Experimentación paralelizada

La paralelización entre los algoritmos GWO, PSO y BA logró resultados notables. Específicamente en las tareas de aprendizaje más grandes y lentas durante la ejecución serializada, estas son las tareas de entrenamiento del modelo de 20 moléculas con función bidimensional de 100,000 registros, mostradas en las figuras 4.3, 4.5 y 4.7. En el que se registró un tiempo promedio de entrenamiento 97.4 % menor respecto a su instancia serializada. Además, en contraste con el entrenamiento por algoritmo de gradiente, se logró una reducción de tiempo aproximadamente de 98.8 %.

A través de todas las iteraciones mostradas en las tablas de resultados A.11 y A.12, A.13 y A.14 así como en A.15 y A.16, es posible observar que el aumento de dimensiones ya no representó un impacto tan considerable como lo fue en los entrenamientos en serie. Aproximadamente el tiempo aumentó entre 1.8 % y 2.22 % en promedio entre la misma arquitectura de modelo con el mismo algoritmo pero distinta cantidad de dimensiones en la función sobre la que se realizó la regresión. Entre el algoritmo BA probado en una y dos dimensiones, figuras 4.3 y 4.2, se muestra una diferencia constante de aproximadamente un segundo en las pruebas de 100, 000 registros. Esta diferencia se presenta como un tiempo de entrenamiento menor en la función bidimensional, por lo que parecería extraña por si sola. Por lo anterior es necesario tomar en cuenta la desviación estándar

TABLA 4.2: tabla comparativa de resultados paralelizados. Muestra las mediciones recogidas en la experimentación bidimensional y con la mayor cantidad de moléculas y registros. Es el conjunto experimental más exigente.

Comparativo de resultados paralelizados					
Reg_Number	**Algorithm**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
100000	BA	46.2261	0.097526743	12.50311029	0.47102672
	GWO	44.0297	0.225786551	1.562495959	0.883936299
	PSO	44.3637	0.090169039	0.434069824	0.44001899
	PFO	43.5416	0.158522484	0.706909573	0.003842571

calculada, en la que se aprecia que los resultados no fueron estables con una variabilidad de 0.65 segundos en el caso de BA en una dimensión. sin embargo, no se observan diferencias sustanciales en experimentos con menor cantidad de observaciones. El mismo fenómeno, aunque con diferencias más pequeñas, aparece en los experimentos con más registros en las comparaciones realizadas entre GWO y PSO en las gráficas 4.5 con 4.4 y 4.7 con 4.6 respectivamente.

En el extremo opuesto de las pruebas, con los modelos de 3 moléculas y tan solo 1000 registros, podemos hacer una comparación interesante. El desempeño en términos de tiempo mostrado entre los algoritmos serializados y paralelizados presenta una reducción de entre 60 % y 65 % entre los pares experimentales homólogos.

Comparando los resultados anteriores contra los resultados de PFO en las tablas A.17 y A.18, se encontró que aunque el algoritmo comienza siendo considerablemente más lento que todos los demás, acercándose más al tiempo de una ejecución serializada, al incrementar la cantidad de registros (manteniendo número de moléculas y dimensiones) el tiempo de entrenamiento máximo es menor comparado contra los otros algoritmos. Siendo PFO el algoritmo con el tiempo de entrenamiento más bajo en el conjunto experimental más demandante mostrado en la tabla 4.2. Al realizar una comparación de los resultados recopilados en las figuras 4.8 y 4.9, se observa que en los experimentos más pequeños los resultados de tiempo son muy similares, mostrando que en estos casos el algoritmo no se ve afectado ante los aumentos de dimensiones. En los conjuntos intermedios, se aprecia el mismo patrón. El fenómeno anterior no se presenta en el experimento más grande. Respecto al impacto que tiene el aumento de dimensionalidad sobre PFO, se encontró un aumento en el tiempo del 6.8 % aproximadamente entre los experimentos realizados con 20 moléculas y 100, 000 registros, resultados recopilados en las figuras 4.8 y 4.9. Aunque, aún con este aumento superior al de otros algoritmos, los tiempos finales de entrenamiento en esta combinación de variables para el experimento sigue siendo el menor.

4.2. Análisis del error cuadrático medio

En esta sección se presenta el análisis de variaciones en el error cuadrático medio. Para realizar esta comparación hay que resaltar que el desempeño del modelo cambia, dependiendo de la cantidad de puntos que componen a la función objetivo, así como la complejidad de la arquitectura del modelo. Por esta razón, en el siguiente análisis se

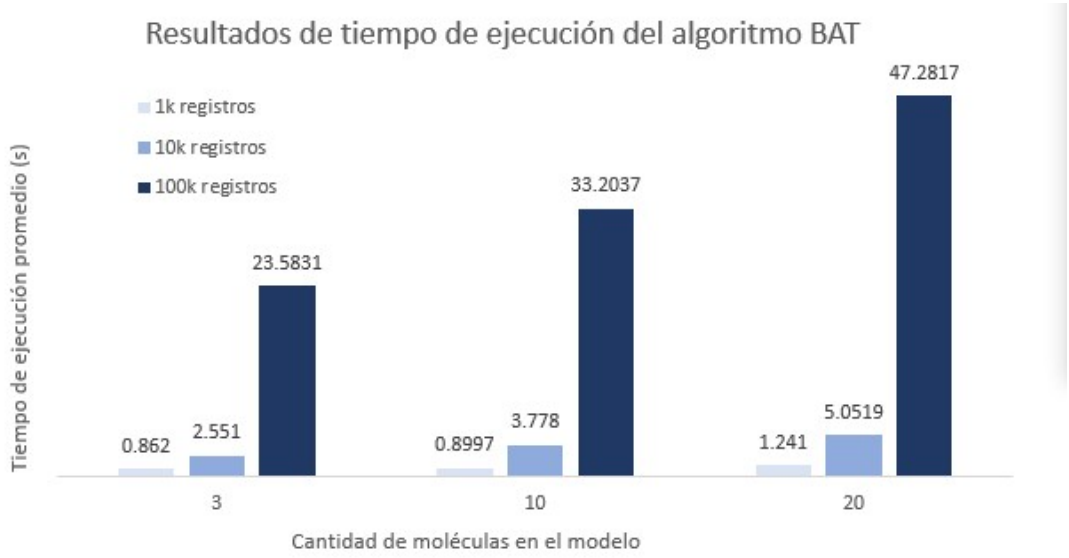

FIGURA 4.2: Gráfica de tiempos de ejecución promedio para el algoritmo paralelizado BA para una función de una dimensión

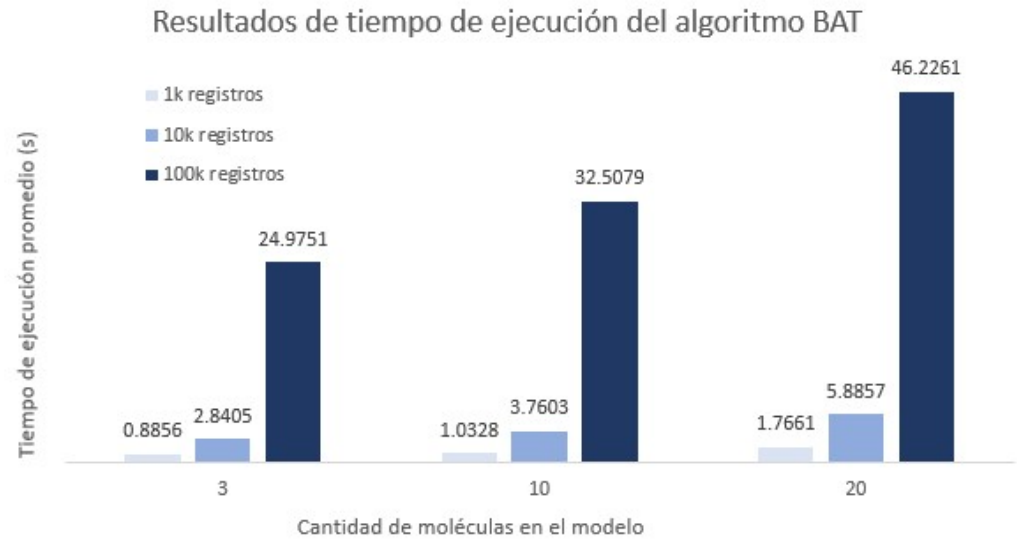

FIGURA 4.3: Gráfica de tiempos de ejecución promedio para el algoritmo paralelizado BA para una función de dos dimensiones

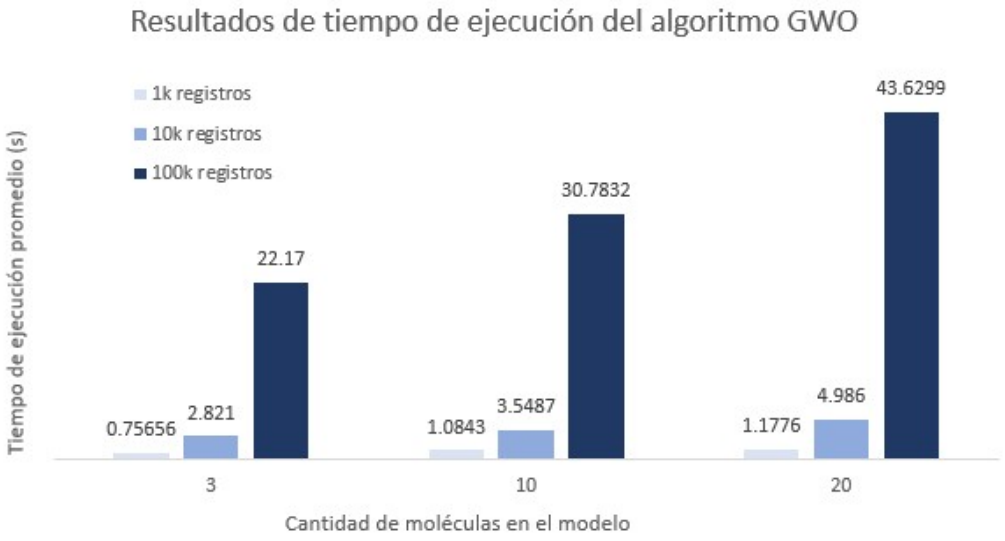

FIGURA 4.4: Gráfica de tiempos de ejecución promedio para el algoritmo paralelizado GWO para una función de una dimensión

presenta primero una comparación aumentando solamente los puntos de composición de la función objetivo. Posteriormente se presentará una comparación relativa al aumento en complejidad del modelo y su impacto en términos de error ante funciones con una mayor cantidad de registros. Cerrando el capítulo, se discutirá el error cuadrático entre algoritmos de entrenamiento.

Esta sección busca identificar las características de escalabilidad de los algoritmos evaluados; así como su desempeño en precisión y convergencia a la función objetivo.

4.2.1. Experimentación serializada

Comenzando con la comparación entre el algoritmo basado en el gradiente, aplicado a una y dos dimensiones en las tablas A.1 y A.2, se presenta el primer problema notable de este método de entrenamiento. La inconsistencia de resultados dificulta la recolección de muestras para el cálculo del error promedio. Además de observarse un aumento constante en el error presente en entrenamientos de modelos con la misma cantidad de moléculas y registros en la función de entrada pero distintas dimensiones. En la mayoría de los casos, el algoritmo reporta un error mayor en su aplicación a la función de dos dimensiones. También es importante resaltar que los resultados de error presentan una desviación estándar de orden bajo, indicando poca varianza en los resultados.

En las comparaciones entre el desempeño de los algoritmos metaheurísticos, modificando solamente la cantidad de dimensiones de la función, se observan distintos fenómenos en cada algoritmo. Comenzando por BA, es notable el aumento del error al entrenar con funciones de mayor dimensionalidad. Este fenómeno se presenta específicamente en todas las arquitecturas del modelo a partir de los 10,000 registros; por ejemplo en el caso de 10 moléculas y 100,000 registros, el error aumenta aproximadamente 10 veces.

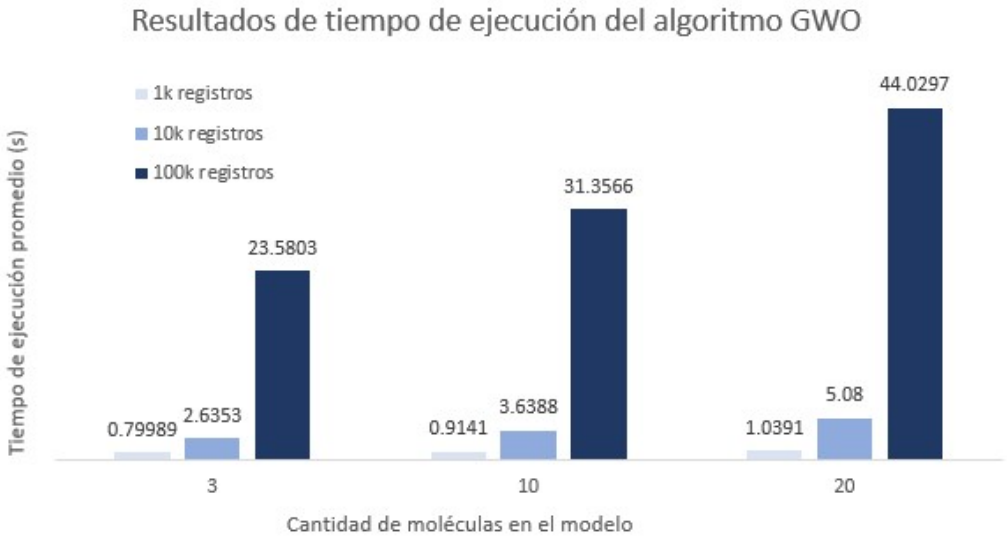

FIGURA 4.5: Gráfica de tiempos de ejecución promedio para el algoritmo paralelizado GWO para una función de dos dimensiones

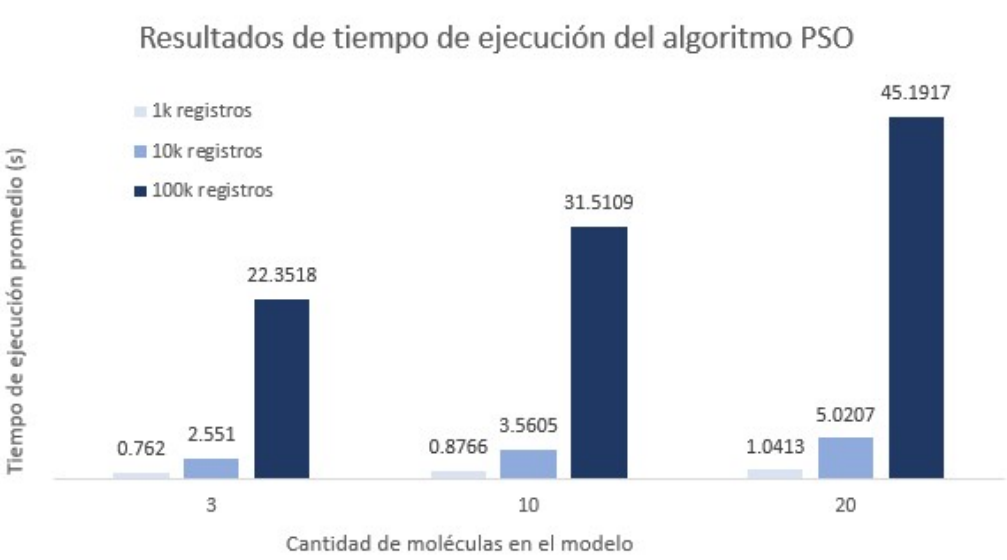

FIGURA 4.6: Gráfica de tiempos de ejecución promedio para el algoritmo paralelizado PSO para una función de una dimensión

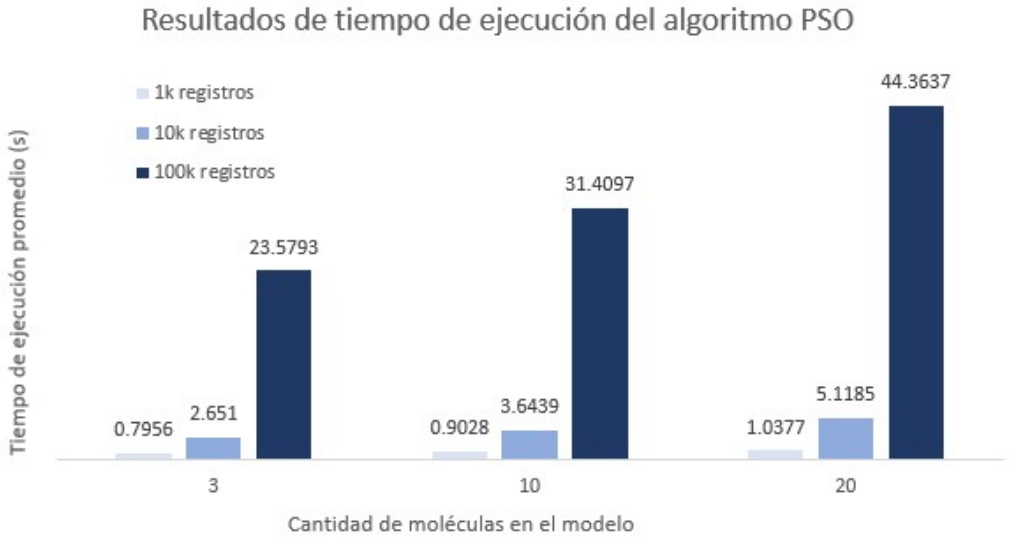

FIGURA 4.7: Gráfica de tiempos de ejecución promedio para el algoritmo paralelizado PSO para una función de dos dimensiones

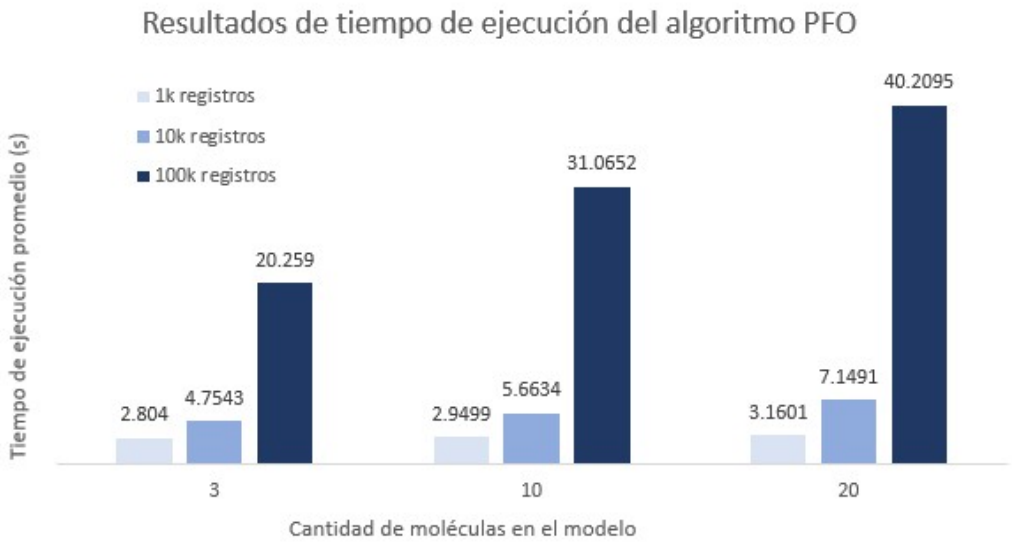

FIGURA 4.8: Gráfica de tiempos de ejecución promedio para el algoritmo paralelizado PFO para una función de una dimensión

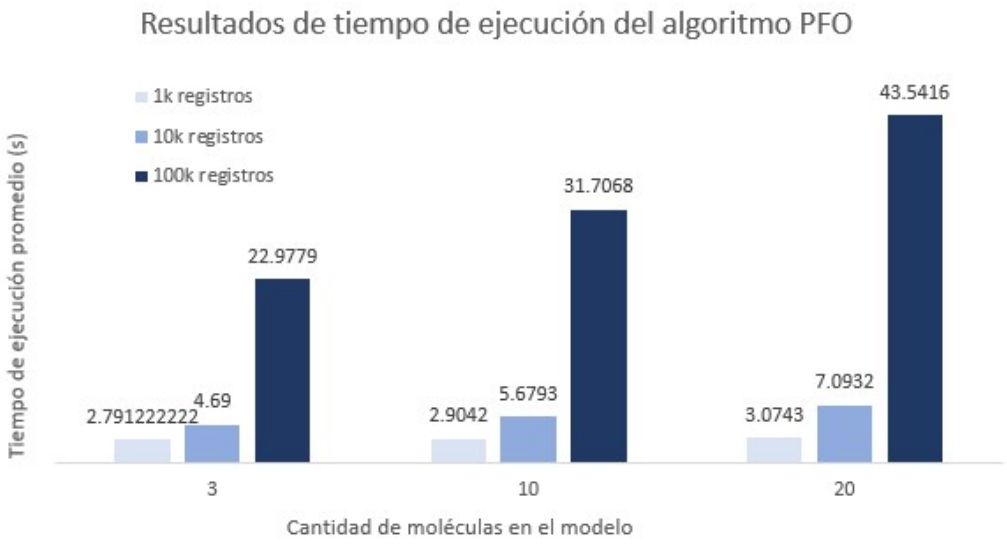

FIGURA 4.9: Gráfica de tiempos de ejecución promedio para el algoritmo paralelizado PFO para una función de dos dimensiones

Esto nos deja ver que el algoritmo BA es muy susceptible a cambios de dimensionalidad; este fenómeno vuelve a ser observado en la experimentación paralelizada, la cual discutiremos en la sección inmediata.

En contraste con los resultados anteriores, el algoritmo GWO presenta mejores resultados al aplicarse en la función bidimensional, sugiriendo su adaptabilidad a problemas con mayores dimensiones. También es interesante observar la reducción del error en los entrenamientos de una dimensión con 100,000 registros. Esto se debe a que al aumentar la complejidad del modelo construido también aumentamos la flexibilidad del mismo para aproximar mejor la función.

Por último, el algoritmo de PSO presenta resultados similares a los encontrados en el algoritmo GWO. Mostrando mejores resultados en la aplicación de entrenamiento en funciones de mayor dimensionalidad. Un resultado interesante en los experimentos unidimensionales de PSO se presenta en las arquitecturas de 10 y 20 moléculas tanto para 10,000 registros como para 100,000 registros, en los que la disminución de error es mínima, mostrando que el algoritmo aplicado no mejorará en gran medida el entrenamiento aunque el modelo aumente en complejidad.

De este análisis se puede concluir que los algoritmos PSO y GWO sugieren mejor desempeño al incrementar las dimensiones de la función objetivo. Además, el algoritmo GWO presenta mejores resultados al ser acoplado al modelo de AHN ya que presenta menor error en la aproximación al aumentar la cantidad de moléculas de aprendizaje.

4.2.2. Experimentación paralelizada

Al referirnos a la Gráfica 4.10, que compara el caso más complejo en la serie experimental, se observa que el error cuadrático medio cambió entre cada algoritmo. Adicionalmente, es importante observar el aumento del error respecto a la misma métrica en la

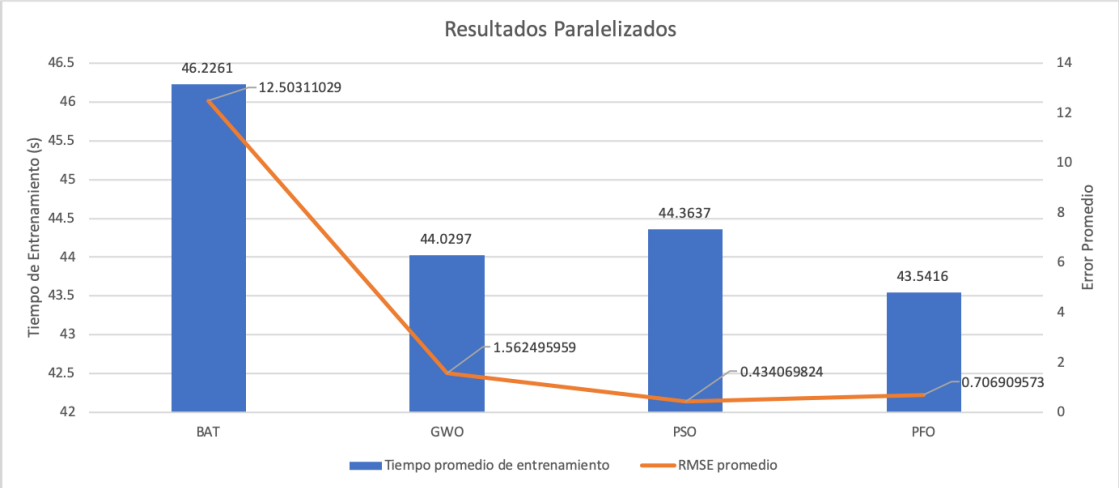

FIGURA 4.10: Gráfica comparativa de resultados en el conjunto experimental más complejo de la experimentación paralelizada

modalidad serializada. El algoritmo BA presenta el mayor error en este caso, aproximadamente 10 veces mayor al algoritmo más próximo (GWO). Este resultado es similar al presente en la ejecución serializada. Respecto a PFO, es notable que además de poseer el menor tiempo de ejecución promedio, es el algoritmo con el segundo menor error, estando sobre el algoritmo PSO solo por 0.3 unidades.

Al centrarnos en la tabla A.18, se observa el aumento progresivo del error cuadrático medio conforme al incremento en número de registros que componen cada función. Aunque es importante que el fenómeno contrario, es decir la progresiva disminución del error, sucede al incrementar la cantidad de moléculas en el modelo de aprendizaje automático. La observación anterior también se repite en la tabla A.17, que presenta la experimentación unidimensional.

Para comparar el desempeño entre los algoritmos estudiados, fueron seleccionados los conjuntos experimentales mas complejos para cada algoritmo, condensados en la tabla 4.2 y graficados en la figura 4.10. De la figura anterior, posible observar que el algoritmo BA mantuvo el error más alto, aproximadamente 28 veces mayor al algoritmo PSO que consiguió el error más bajo. PFO fue el segundo algoritmo con el error mas bajo, solamente 0.3 unidades por encima de PSO. Sin embargo, PFO resulto tener la desviación estándar mas baja entre todas las pruebas, sugiriendo resultados mas consistentes de entre las pruebas.

4.3. Caso de estudio: aplicación de PFO y AHN en HAR

En esta Sección se presentan los resultados del caso de estudio principal. La aplicación al reconocimiento de actividades humanas del algoritmo AHN entrenado utilizando el algoritmo PFO paralelizado localmente. En todos los experimentos realizados se mantuvo la misma estructura de 20 moléculas totales y 100 pirañas con 3 presas.

TABLA 4.3: tabla de representación inicial de datos en la base de entrenamiento.

Categoría	Cantidad de Registros	% del total
1	153	0.539568345
2	152	0.536041755
3	187	0.659472422
4	152	0.536041755
5	186	0.655945832
6	5324	18.77556778
7	6148	21.68147835
8	5344	18.84609959
9	200	0.705318098
10	2551	8.996332346
11	7817	27.56735788
20	142	0.50077585
Total	28356	100

La base de datos "UP-Fall Detection Dataset" [29] será utilizada sin características de visión; es decir no contendrá imágenes. El estudio con imágenes requiere de técnicas de pre procesamiento y refinación que podrán ser exploradas en trabajos futuros.

El conjunto de datos utilizado consta de 868 distintas dimensiones o variables, que se traducen en el mismo número de dimensiones para el espacio de búsqueda y entrenamiento. "UP-Fall Detection Dataset" [29] contiene 28,357 observaciones con un total de 12 categorías como se muestra en la tabla 4.3. El conjunto de datos original contiene una categoría que se ha excluido de este ejercicio; la categoría con el nombre 20. La razón es que esta categoría existe solamente para 142 registros y es en realidad una categoría dedicada a observaciones no clasificadas o nulas.

La tabla 4.4 ofrece una explicación cualitativa de la actividad realizada en cada categoría, y servirá para explicar algunos de los resultados más adelante. Es importante decir que las categorías 1, 2, 3, 4, 5 y 9 sufren de una muy baja representación en la base de datos; es decir que existe un desequilibrio que puede dificultar la clasificación de categorías sub representadas. Para solucionar el desequilibrio existen varias opciones, una de ellas es unificar todas las categorías en una sola, también sería posible utilizar una técnica de generación de registros sintéticos. El mayor problema con la primera opción es que no es conveniente unificar tantas categorías ya que podrían ser extremadamente distintas entre ellas, lo que provocaría en la unificación una categoría sintética que sea extremadamente dispersa en el espacio de observaciones. Esto es verificable en problemas de baja dimensionalidad pero en este ejercicio nos enfrentamos a más de 800 dimensiones. La segunda opción, generación sintética de registros, se dificulta al considerar que las categorías en cuestión representan alrededor del 0.5 % de la base de datos cada una. Los registros sintéticos tendrían que basarse en muy pocas observaciones para lograr nivelar la representatividad; lo que podría conducir a un marcado sesgo en la clasificación.

Se decidió utilizar una técnica de generación de registros sintética ya que la única otra

TABLA 4.4: Explicación de categorías en la base de datos "UP Fall Detection"

Actividad	Descripción
1	Caída delantera usando las manos
2	Caída delantera usando rodillas
3	Caída hacia atrás
4	Caída lateral
5	Caída por silla vacía
6	Caminata
7	De pie
8	Sentado
9	Recogiendo un objeto
10	Saltando
11	Recostado

TABLA 4.5: tabla de representación de categorías en la base de datos después de SMOTE.

Categoría	Cantidad de Registros	% del total
1	1836	4.542193414
2	1976	4.888548032
3	2431	6.014200539
4	1976	4.888548032
5	2418	5.982039039
6	5324	13.17137132
7	6148	15.20991564
8	5344	13.22085055
9	2600	6.432300042
10	2551	6.311075926
11	7817	19.33895747
Total	40421	100

alternativa viable sería dejar de lado 6 de las 10 categorías analizadas, simplificando el problema más allá de lo necesario para probar el desarrollo realizado. Utilizando la técnica "Synthetic Minority Oversampling Technique" [7], o por sus siglas, SMOTE incrementamos la cantidad de registros de las 6 categorías discutidas hasta alcanzar un porcentaje de representación aceptable.

La técnica SMOTE genera nuevas observaciones sintéticas [7] introduciendo pequeñas variaciones en cada dimensión del espacio, tomando como punto inicial un punto que pertenezca a la categoría a balancear. Genera entonces vecinos sintéticos para cada grupo. La base de datos resultante de este proceso se muestra en la tabla 4.5. En esta tabla resultante se muestra el aumento de representación por categoría, logrando que cada una de ellas tenga alrededor de 4.8 % de representación en la base.

TABLA 4.6: Resultados de los cinco ejercicios de entrenamiento.

ET_Mean	ET_stDev	Moléculas	RMSE_Mean	RMSE_stDev
16972.75	255.1953	20	0.234629	0.0261216

Por lo tanto la base de datos utilizada se compone de 40, 421 observaciones y 11 categorías. Partiendo de este conjunto, realizaremos una selección aleatoria de registros para ser utilizados como conjunto de pruebas; el conjunto restante será utilizado para la tarea de entrenamiento del modelo. La división se realizó de acuerdo a una selección aleatoria y con barajado previo. se dedico el 80 % de la base al entrenamiento y 20 % para pruebas del modelo, alocando 32,336 registros y 8,040 registros respectivamente.

Es importante remarcar que las variables de la base de datos son numéricas y continuas en todos los casos; no existen variables categóricas que requieran codificación ni datos vacíos que demanden un proceso de rellenado sintético. Por esta razón, solamente fueron realizadas dos transformaciones a la base de datos, primero la codificación de las 11 categorías de clasificación utilizando el método "one-hot" [44]; resultando en 11 variables de salida. Este método de codificación construye una columna correspondiente a cada categoría, colocando un 1 a la columna correspondiente y rellenando con 0 las demás. La segunda transformación realizada fue un proceso de normalización para todas las variables, manteniendo los valores entre 0 y 1 se logra que no exista un sesgo artificial en el ajuste del modelo hacia variables con valores de mayor tamaño. Este proceso de tratamiento es común en cualquier ejercicio de aprendizaje automático, por lo que no presenta una ventaja especial hacia este modelo en particular [21]. Adicionalmente el proceso para la clasificación multivariada busca que el modelo entregue como resultado una probabilidad de pertenencia para cada categoría, seleccionando como etiqueta final la categoría más probable.

Fueron realizados cinco ciclos completos, esto incluye la división aleatoria en conjunto de entrenamiento y de prueba, el proceso de entrenamiento y la predicción con el modelo exportado. De cada experimento se registro el tiempo total de entrenamiento; también, utilizando el vector de pertenencias con probabilidades antes de la asignación de categorías, el error cuadrático medio de la predicción. Finalmente utilizando las clasificaciones finales, se calcularon otras métricas de clasificación que se presentarán y discutirán más tarde en esta misma sección.

En la tabla 4.6 se muestra que el tiempo de entrenamiento promedio fue de 16663.25 segundos, aproximadamente 4.71 horas. Resultado que muestra nuevamente la velocidad de entrenamiento alcanzada por la paralelización junto con las aproximaciones realizadas por el algoritmo PFO. Adicionalmente, la desviación estándar de alrededor de 255 segundos, aproximadamente 4 minutos, representando solamente 1.5 % del tiempo de entrenamiento. La métrica del error cuadrático medio en este caso no es la adecuada para juzgar los resultados de la clasificación. Un error menor podría indicar que, en general, el modelo predijo una alta probabilidad de pertenencia a una categoría que de hecho resulto ser correcta. También podría indicar que no existieron muchos casos en los que la clasificación, según la probabilidad de pertenencia estimada, fuera errónea; ya que la diferencia entre una probabilidad predicha alta contra un 0 en ese componente del vector, aumentaría el error cuadrático. Pero esta métrica no es definitiva, por lo que

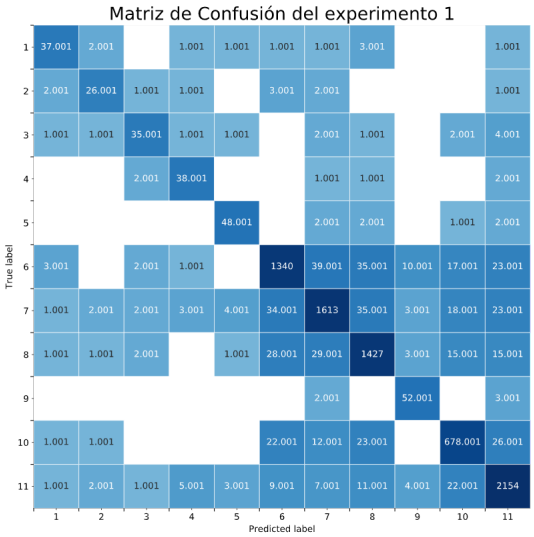

FIGURA 4.11: Matriz de Confusión del experimento 1

se construyó una matriz de confusión utilizando los conjuntos de predicciones y de la cual se calculan las métricas finales del clasificador.

La matriz presentada en la figura 4.16 es en realidad la matriz promedio de los resultados de todos los experimentos realizados. Esta es la matriz en la que concentraremos la discusión ya que sirve como condensado del comportamiento del clasificador; para ejemplos específicos se hará referencia a las figuras 4.11, 4.12, 4.13, 4.14 y 4.15. Es posible observar como la mayor concentración de clasificaciones equivocadas se presenta entre las categorías 6, 7 y 8; existen varias explicaciones posibles. Hay que recordar que en estas tres categorías se encuentra una gran cantidad de registros de la base, según la tabla 4.5 estas tres conforman el 40 % del universo de observaciones; por lo que es posible que existan más predicciones erróneas al existir también más registros que evaluar. También es posible que estas tres categorías sean similares entre ellas, provocando que el clasificador no pueda diferenciar fácilmente entre ellas, lo que podría explicar por qué la mayoría de los errores de predicción se presentan dentro de este conjunto. Este último punto es de naturaleza cualitativa y verificando la actividad en la tabla 4.4 proveniente de [29], encontramos que la categoría 6 y 7 son próximas al tratarse de una caminata y pararse de pie respectivamente. Respecto a la categoría 8, solamente podría explicarse su relación con la caminata por la similitud del movimiento registrado. Esto puede ser mejorado al modificar las ventanas de tiempo para cada observación.

Lo discutido en el párrafo anterior no es suficiente para una evaluación conclusiva de un modelo de clasificación, por lo que a continuación, en la tabla 4.7, se presentan las métricas de exactitud, precisión y la puntuación F1; métricas seleccionadas en el Capítulo 3.

En el caso de prueba presentado observamos una exactitud promedio de aproximadamente 94 %, lo que indica que el modelo entrenado se acerco a la distribución total de registros en ese porcentaje. Se observa en su respectiva desviación estándar que no existe gran variación entre experimentos, indicando que el modelo entrenado es estable y

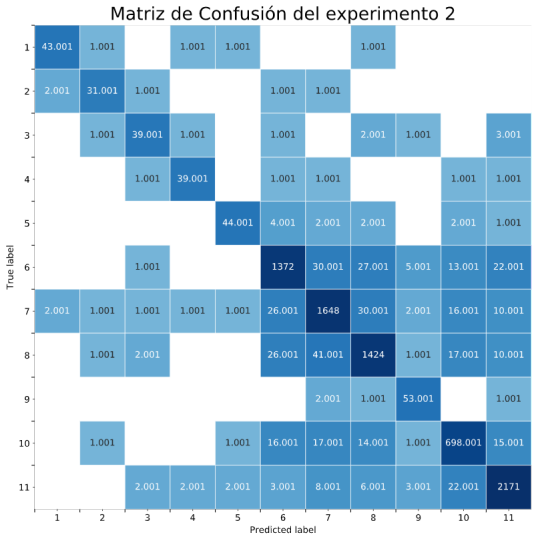

FIGURA 4.12: Matriz de Confusión promedio del experimento 2

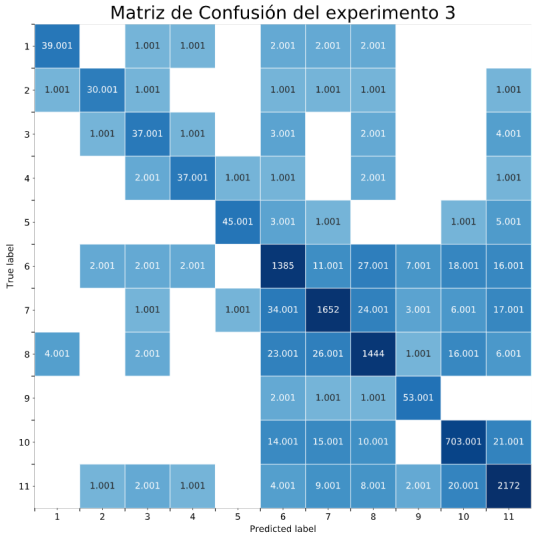

FIGURA 4.13: Matriz de Confusión promedio del experimento 3

TABLA 4.7: Métricas de evaluación del modelo en los resultados experimentales.

Métrica	Exactitud/Accuracy	F1	Precisión
Experimento 1	0.931476	0.865363	0.856442
Experimento 2	0.945653	0.908549	0.908788
Experimento 3	0.950006	0.902554	0.910111
Experimento 4	0.938440	0.889506	0.898514
Experimento 5	0.944907	0.897315	0.897365
Promedio	0.942096	0.892657	0.894244
Desviación estándar	0.007233	0.016786	0.021910

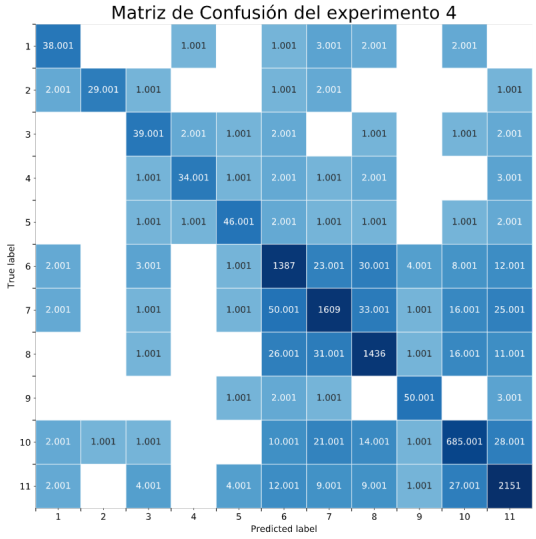

FIGURA 4.14: Matriz de Confusión promedio del experimento 4

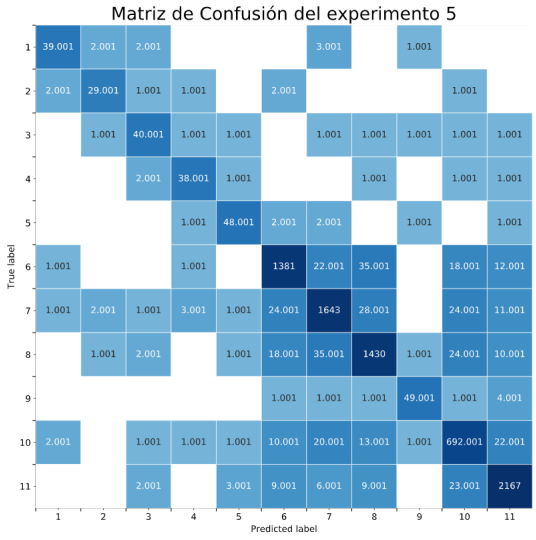

FIGURA 4.15: Matriz de Confusión promedio del experimento 5

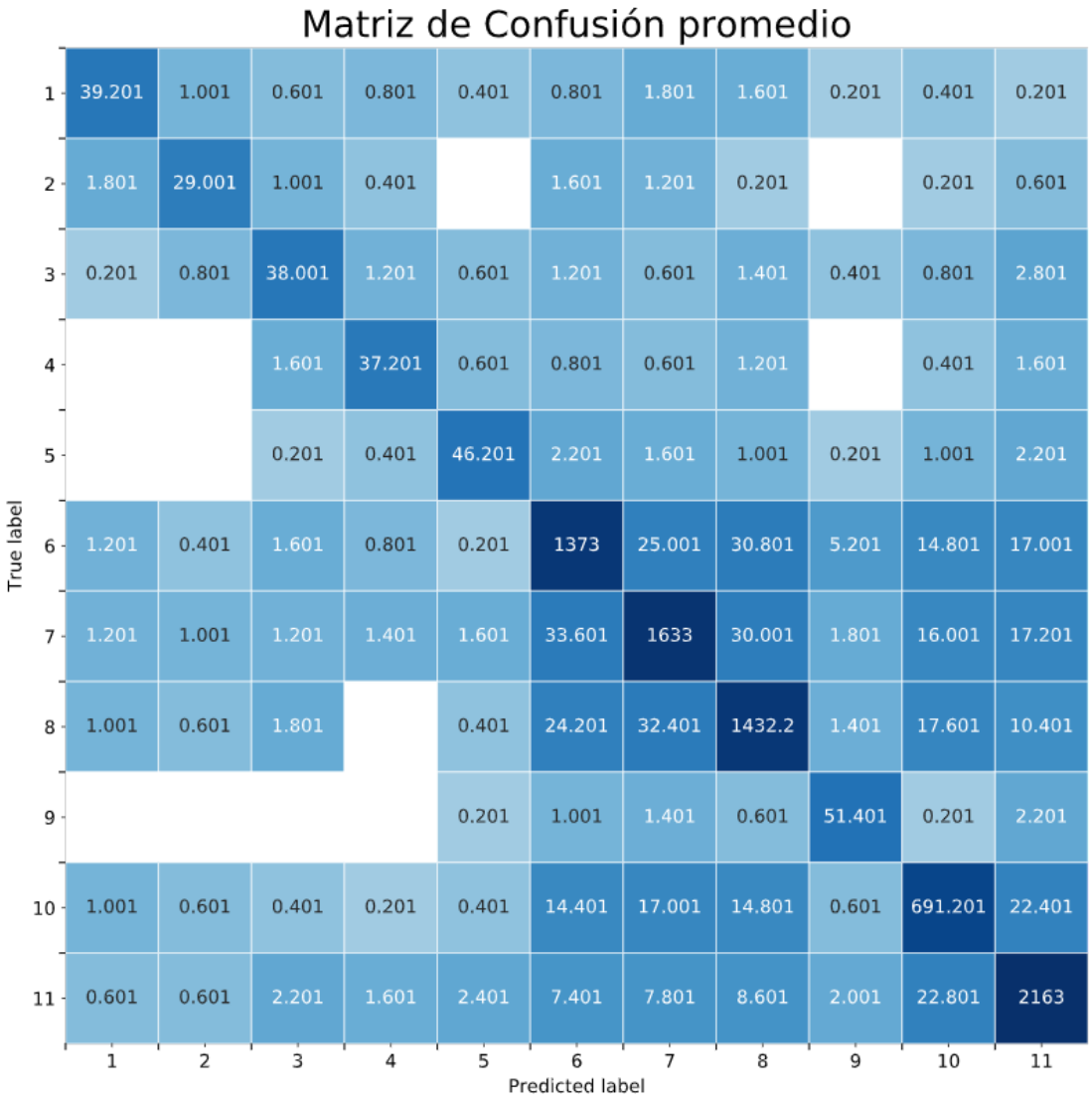

FIGURA 4.16: Matriz de Confusión promedio

consistente en su comportamiento. Este es un resultado muy favorable ya que sugiere que el presente desarrollo es confiable y no presentará divergencias en resultados de tareas similares.

En el ejercicio discutido, la precisión promedio fue de 0,89, por lo que se puede decir que el modelo predijo 89 % de las observaciones de prueba de forma correcta. Calculando la desviación estándar obtenemos que fue de 0.0167, apuntando nuevamente a la estabilidad de los resultados. Es muy importante notar que en el "Experimento 1" se encontró una precisión más baja en relación a las demás observaciones; es posible que este resultado se deba a variaciones en la estrategia de búsqueda en el algoritmo PFO, a la aleatoriedad implementada en el modelo de AHN o a una combinación de ambas.

La puntuación F1 promedio en el caso de estudio es de 0.8926, mostrando una alta capacidad de flexibilidad del modelo AHN. Adicionalmente, del segundo al quinto experimento se confirma la estabilidad de resultados. Nuevamente el primer experimento mostró una diferencia considerable, que puede ser atribuida al mismo efecto de aleatoriedad en condiciones iniciales o de exploración del algoritmo PFO y AHN. Esta métrica también indica que el modelo resultante es capaz de clasificar las observaciones de forma correcta a la par de minimizar los casos falsos negativos respecto a su pertenencia; en el caso de un problema multiclase, esto quiere decir que minimiza las clasificaciones traslapadas erróneamente, adaptándose favorablemente a clases similares respecto a su distancia en el espacio. Todas estas métricas podrían mejorar con ajustes a los hiperparámetros del modelo y del algoritmo de entrenamiento.

4.4. Discusión y resultados del algoritmo PFO

El algoritmo presentado en esta investigación, contrastado en la tabla 4.8 contra los demás algoritmos discutidos, incorpora dos criterios lógicos que fomentan la exploración en puntos de interés, además de incrementar la exploración de nuevas áreas en el espacio a través de un sistema de penalizaciones históricas en puntos de interés. El primer incentivo a la exploración de puntos de interés es la inclusión de una función de movimiento distinta cuando un agente se encuentra a cierta distancia de un punto de interés; esta función de movimiento simula el desplazamiento en espiral de un agente en el cardumen hacia alguno de los puntos de interés encontrados hasta ese momento. El sistema de penalizaciones históricas consiste en la reducción progresiva de la distancia a la que un agente debe encontrarse del punto de interés para modificar su comportamiento de navegación en espiral a un desplazamiento lineal hacia el centro de un punto de interés; esta distancia se reduce progresivamente según un hiperparámetro seleccionado. De esta forma, los agentes buscarán explorar regiones de interés pero evitando la sobre explotación histórica.

Los criterios de actualización del conjunto de soluciones, es decir, los métodos para mover las soluciones posibles en el espacio se pueden realizar con distintas reglas; aunque la función de movimiento sea la misma. Esto puede impactar negativamente en la exploración de una función ya que no existe diferencia de movimiento cuando una solución se encuentra cerca de un punto de interés o lejos del mismo. La falta de control significa que movimientos o mutaciones bruscas cerca de puntos de interés pueden provocar

que el algoritmo no converja en una solución o tarde demasiadas iteraciones en hacerlo. En contraste, si el algoritmo es configurado con movimientos demasiado sutiles, la exploración será lenta y el espacio explorado muy limitado. Por esta razón, el algoritmo propuesto incorpora dos criterios de exploración, social e individual, así como dos funciones de movimiento. Estas funciones de movimiento serán aplicadas según la posición de la solución, si se encuentra cerca de un punto de interés se moverá de forma lineal y controlando su velocidad mientras que si se encuentra fuera del rango de interés, se moverá según un espiral logarítmico, buscando acercarse a puntos de interés desde distintas posiciones en el espacio y explorando la mayor área posible.

Por último, el algoritmo PFO incorpora un parámetro de control que permite configurar la cantidad de puntos de interés permitidos durante la exploración. Estos puntos de interés son considerados presas y son explorados según los criterios anteriores. Esta capacidad es importante ya que limitar los puntos de interés permitidos a uno solo como es el caso de PSO, puede sesgar su exploración. Además, permitir que el conjunto de soluciones se agrupe y explore los alrededores de múltiples puntos de interés, aumenta las probabilidades de encontrar mejores soluciones cercanas a los puntos previamente categorizados. Es importante decir que esta capacidad podría ser perjudicial para algoritmos que no cuenten con mecanismos que los preparen para incentivar nuevas áreas exploradas.

TABLA 4.8: tabla comparativa de características de interés en algoritmos metaheurísticos de referencia y el algoritmo PFO

Algoritmo	Parámetros ajustables	Incentivo a exploración	Castigo de sobre-explotación	Funciones de movimiento	Control sobre puntos de interés
PSO	6	No	No	1	No
GWO	1	No	No	3	No
BA	5	No	No	1	No
PFO (propuesto)	5	Si	Si	2	Si

Respecto a la implementación general de la propuesta, la paralelización mostró resultados favorables, reduciendo el tiempo necesario para entrenar el modelo sin sacrificar exactitud respecto a su versión serializada. Esto se debe a que en realidad las operaciones realizadas no están cambiando, simplemente son administradas de otra forma para ser calculadas de forma más eficiente.

Después de recoger los resultados anteriores, es importante discutir las ventajas y desventajas del algoritmo PFO como motor de entrenamiento de AHN. La principal ventaja observada tanto en las pruebas serializadas y paralelizadas fue que en los casos con arquitecturas de AHN más grandes, consistentemente consiguió tiempos de entrenamiento menores a los de los demás algoritmos. Terminando con un tiempo de ejecución aproximadamente 10 % menor al de los demás. Esto muestra la capacidad del algoritmo de crecer en volumen, adaptándose a la complejidad del modelo a entrenar. Esta ventaja es también pie de la primera desventaja del algoritmo, ya que pudimos observar que en pruebas más reducidas, donde se entrenaba sobre funciones de menos dimensiones, observaciones y con arquitecturas más pequeñas, el algoritmo fue en todos los casos más lento. Por esta razón el algoritmo PFO en su estado actual parece no ser la mejor opción para problemas de menor escala. Respecto al error promedio en comparación a los otros algoritmos, PFO mostró un error de 0.7 en las pruebas paralelizadas y se colocó como el

TABLA 4.9: Taabla de resultados del artículo "UP-Fall detection dataset" [29] y de la propuesta presentada

Métrica	Exactitud/Accuracy	F1	Precisión
Artículo "UP-Fall detection dataset"	0.9326	0.7351	0.6819
Propuesta	0.9420	0.8926	0.8942

segundo algoritmo con menor error reportado. Esto representa una ventaja considerable frente a los demás ya que ofrece un buen balance entre exactitud y tiempo de ejecución.

La capacidad de mantener los resultados de error consistentes con los niveles de los demás algoritmos radica en la integración de las dos políticas de movimiento distintas y a la estrategia de incentivos y castigos a la sobre explotación. Las dos funciones permiten un movimiento más variado según la posición relativa del agente y por lo tanto la exploración se da con mayor sensibilidad cerca de puntos de interés. La capacidad de escalamiento del algoritmo PFO se debe a que al aproximarse a una función más compleja que requiere de más iteraciones, poco a poco las pirañas exploran más con la política sencilla de ataque lineal, provocando que cada vez menos pirañas requerían recalcular su distancia a todos los puntos de interés; efectivamente disminuyendo la complejidad y cantidad de cálculos conforme pasa el tiempo. En un problema demandante de recursos esta característica provoca que el aumento gradual en tiempo sea menor.

Comparando de los resultados obtenidos contra los encontrados en el articulo por Martínez-Villaseñor y Ponce [29] encontramos que en este artículo fueron presentados algunos modelos de clasificación para la base de datos "UP Fall Detection", entre los que destaca por sus resultados y su similitud con el modelo presentado en este trabajo el uso de una red multicapa de perceptrones clásicos. También es presentada una red neuronal convolucional pero es utilizada sobre la base de datos extendida con imágenes, extensión que no está en el alcance de este trabajo. En el artículo se encontraron que los mejores resultados de exactitud, precisión y F1 para las redes neuronales sin uso de cámaras fueron de 93,26 %, 73,51 % y 68,19 % respectivamente. Existe una mejora en todas las métricas al comparar estos resultados contra los obtenidos en la tabla 4.7; la comparación con los resultados de la propuesta se muestra en la tabla 4.9. Esta mejora se debe a la aptitud del modelo de AHN para la tarea de clasificación aplicada, como mostró Ponce [40].

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

El problema del reconocimiento de actividades humanas es un tema de investigación abierto, con aplicaciones médicas, de entretenimiento y seguridad. Este problema comúnmente implica una gran cantidad de datos que no son manejables para modelos estadísticos convencionales, dada su complejidad dimensional y volumen. En la implementación del modelo AHN previa a este desarrollo era utilizado el algoritmo de optimización basado en el gradiente de la función objetivo para su entrenamiento. Lograr un entrenamiento eficaz y estable con este método era difícil por su alto costo computacional, que se veía traducido en altos tiempos de procesamiento y poca estabilidad.

Lo anterior nos conduce a buscar soluciones que incorporen nuevos modelos de aprendizaje automático que resuelvan el problema dimensional, pero comúnmente se encuentran implementados con estrategias de procesamiento que no favorecen su escalabilidad. Es por esta razón que fueron explorados algoritmos metaheurísticos que busquen una solución óptima aproximada para el problema de ajuste interno del modelo. En este desarrollo fueron utilizados los algoritmos metaheurísticos BA, GWO y PSO. El algoritmo BA está inspirado en el comportamiento de navegación de los murciélagos y sus capacidades de ecolocación [66] [64]. El optimizador GWO o "Grey Wolf Optimizer" fue modelado a partir de los patrones de caza de las manadas de lobos grises; incorporando también un sistema de jerarquización de los individuos [33]. Por último el algoritmo "Particle Swarm Optimization" modela la navegación de una parvada de pájaros, moviéndose según patrones sociales e individuales [18]. Adicionalmente, el desarrollo inició con la implementación de una estrategia de entrenamiento utilizando el algoritmo de optimización por gradiente.

Los algoritmos de optimización metaheurística presentan problemáticas como la poca cantidad o el exceso de hiperparámetros y la falta de políticas de movimiento de agentes que ayuden a configurar la estrategia de búsqueda cuando se busca utilizarlos como motores de entrenamiento. Lo anterior conjunto al desarrollo de un nuevo algoritmo metaheurístico llamado Algoritmo de Festín de Pirañas o PFO por sus siglas en inglés. Este algoritmo fue diseñado con 5 hiperparámetros para permitir la configuración de la estrategia de búsqueda, una política de incentivos a la búsqueda y de castigos a la sobre exploración para evitar la enajenación con mínimos locales, dos funciones de descripción de movimiento para evitar el posible sesgo de rutas; acelerando la búsqueda en puntos de interés a la par de no detener la exploración en otras áreas, y por último una

política de control de puntos de interés. Esta última otorga el control de la exploración a un hiperparámetro que modifica radicalmente el comportamiento de la búsqueda. Por último se incorporó una estrategia de procesamiento paralelo con el fin de acelerar el entrenamiento del modelo manteniendo los resultados de una estrategia serializada. La combinación del algoritmo PFO para el entrenamiento del modelo de AHN es la principal contribución de esta investigación.

La incorporación de algoritmos metaheurísticos, en ejecución serializada, mostró una reducción inmediata en el tiempo de entrenamiento a casi el 50 % en todos los casos, a la par de mejorar los resultados de error cuadrático medio respecto a un algoritmo de entrenamiento clásico basado en el gradiente, mostrando la efectividad de los algoritmos metaheurísticos y su comportamiento con el modelo AHN.

Posteriormente la integración del procesamiento paralelo llevó el tiempo de entrenamiento a un máximo de aproximadamente 40 segundos en una prueba de 20 moléculas de AHN y una función bidimensional de 100,000 registros. Misma configuración que inicialmente, con el algoritmo gradiente serializado, tomó más de 3500 segundos.

La comparación entre PFO y los demás algoritmos probados, resultó en que PFO es capaz de lidiar con aumentos dimensionales de mejor forma, ya que mantuvo el error cuadrático medio en un rango similar al de los demás pero con aumentos menores en tiempo de ejecución al incrementar las dimensiones de la función de prueba.

Por último, el caso de prueba de HAR, mostró el desarrollo completamente integrado, con un tiempo de entrenamiento promedio de 4.7 horas. También mostró una puntuación F1 de 0.89, mostrando un muy buen balance entre precisión y cobertura. Esta métrica sirve para mostrar la flexibilidad del modelo AHN y cómo el algoritmo PFO es capaz de mantener los resultados de la optimización consistentemente.

Este desarrollo fue documentado en 1625 pruebas experimentales que tomaron alrededor de 78 horas de procesamiento en su conjunto.

En este trabajo se presentó un desarrollo nuevo para el problema de reconocimiento de actividades humanas que incorpora una estrategia de procesamiento paralelo, la integración de un nuevo algoritmo metaheurístico inspirado en la naturaleza como estrategia de entrenamiento para el modelo de Redes de Hidrocarbonos Artificiales, mostrando resultados favorables en flexibilidad del modelo clasificador y algoritmo de entrenamiento. También abriendo las puertas a más casos de estudio que utilicen este desarrollo, no sólo para reconocimiento de actividades humanas.

5.2. Trabajo futuro

Primero la construcción de una estrategia de procesamiento distribuida. Esto con el fin de habilitar la capacidad de probar este desarrollo en bases de datos mucho más grandes y documentar el comportamiento del mismo ante una arquitectura distribuida. La implementación propuesta integrará el algoritmo PFO y el modelo AHN en un sistema de archivos y distribuidos, donde se separarían las tareas de aprendizaje sobre secciones de una base de datos, delegando el procesamiento del entrenamiento a nodos trabajadores. Esto podría traer mejoras en tiempo de ejecución, abriendo las puertas a casos de uso más complejos. De la mano con el punto anterior, una de las razones por la que se

implementaron los algoritmos paralelizados utilizando Numba, fue por su integración con CUDA; facilitando las modificaciones necesarias para lograr la implementación del trabajo realizado en una GPU, acelerando aún más el proceso.

El segundo punto es la modificación de las funciones de actualización y la comparación del algoritmo PFO modificado contra otros optimizadores. Especialmente la ecuación de la degradación espiral puede mejorarse para aumentar la sensibilidad de la búsqueda. Esto va de la mano con la realización de un estudio de pruebas de optimización formales para el algoritmo PFO, comparándolo contra otros algoritmos de optimización en funciones estandarizadas.

Adicionalmente, se propone realizar mejoras al código implementado para disminuir la cantidad de dependencias, parametrizar variables de entrada para lograr la generalidad de la herramienta. Esta propuesta es especialmente importante para incrementar el impacto del desarrollo, convirtiéndolo en una aportación al campo.

Sobre las métricas y comportamiento, la primera mejora propuesta es la inclusión de la métrica de pérdida de Hamming o "Haming Loss" con el fin de funcionar como punto comparativo entre modelos de aprendizaje automático en los que el algoritmo PFO sea implementado como motor de entrenamiento. En continuación a la sugerencia anterior, está la comparación de más implementaciones de PFO y modelos de aprendizaje automático realizando la misma tarea de clasificación o regresión.

Como recomendación final, es importante realizar más experimentos en el caso de prueba actual para identificar mejores parámetros iniciales de configuración. De esta forma es posible mejorar el desempeño obtenido, así como confirmar tendencias generales que no son verificables con tan pocas observaciones experimentales.

Capítulo 6

Contribuciones

La contribución principal de esta investigación es la combinación del algoritmo PFO y del modelo AHN para lograr el entrenamiento de este modelo en tareas que, por su tamaño, no le eran posibles siendo entrenado con el algoritmo de optimización basado en el gradiente. Las mejoras del entrenamiento no representaron disminuciones en precisión, exactitud o puntuación F1 respecto a pruebas realizadas anteriormente por otros autores.

Este trabajo y sus desarrollos han sido utilizados en una publicación de revista científica y dos capítulos de libro, mostrados a continuación.

1. Artículo de revista científica: Ponce, H., de Campos Souza, P. V., Guimarães, A. J., & González-Mora, G. (2020). Stochastic parallel extreme artificial hydrocarbon networks: An implementation for fast and robust supervised machine learning in high-dimensional data. Engineering Applications of Artificial Intelligence, 89, 103427.
2. Capítulo de libro: Ponce, H., González-Mora, G., Morales-Olvera, E., & Souza, P. (2020). Development of fast and reliable nature-inspired computing for supervised learning in high-dimensional data. In Nature Inspired Computing for Data Science (pp. 109-138). Springer, Cham.
3. Capítulo de libro: Ponce, H., González, G., Miralles-Pechuán, L., & Martínez Villaseñor, M. L. (2017, October). Human Activity Recognition on Mobile Devices Using Artificial Hydrocarbon Networks. In Mexican International Conference on Artificial Intelligence (pp. 17-29). Springer, Cham.

Además el código completo de este desarrollo ha sido publicado en el siguiente repositorio para su consulta por la comunidad:

1. https://github.com/jggmemo/pfo_ahn

En el repositorio se encuentran los códigos de la implementación de AHN en Python, los cuatro algoritmos metaherísticos utilizados en sus versiones serializadas y paralelizadas (nombradas con el sufijo "_func"). También se incluye un código principal de prueba para la tarea de clasificación HAR y dos códigos que generan las pruebas de la función unidimensional y bidimensional.

Apéndice A

Tablas de resultados

TABLA A.1: Tabla de resultados de la experimentación del algoritmo basado en el gradiente en el entrenamiento para una función de una dimensión.

GRA - 1 dimensión - serializado					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	3.8886	0.045127474	0.082046356	2.479796543
	10	7.9948	0.182720673	0.008317947	0.096001027
	20	24.0059	2.727063844	0.001634286	0.334897752
10000	3	58.3654	0.268625803	0.068028387	0.011478893
	10	106.1761	NA	0.013054632	NA
	20	302.1109	3.368751811	1.02112586	0.457717299
100000	3	762.9767	0.45551974	0.061154494	2.49328703
	10	1532.0348	NA	0.129269109	NA
	20	3177.6043	NA	0.173517809	NA

TABLA A.2: Tabla de resultados de la experimentación del algoritmo basado en el gradiente en el entrenamiento para una función de dos dimensiones.

GRA - 2 dimensiones - serializado					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	3.0123	0.115918602	0.020505673	2.002507855
	10	11.9919	16.90617165	0.002405365	1.943855872
	20	34.5625	NA	0.0651249	NA
10000	3	68.3822	0.325250604	0.689884	0.056141421
	10	133.8957	0.40640102	0.665436375	0.478341262
	20	403.5549	NA	0.35341605	NA
100000	3	970.7345	0.628040736	0.706793608	0.513
	10	1816.4367	3.876043087	0.705615141	0.234125903
	20	3519.7253	NA	0.702199543	NA

TABLA A.3: Tabla de resultados de la experimentación serializada del algoritmo BA en el entrenamiento para una función de una dimensión.

BA - 1 dimensión - serializado					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	2.967625	0.120836391	0.106713541	0.054700847
	10	8.79075	0.060990163	0.019182904	0.011108101
	20	17.880625	0.314662518	0.013823617	0.0000146
10000	3	27.3375	0.566599604	0.115127509	0.051988283
	10	81.181875	3.438660277	0.029688675	0.021393917
	20	160.31825	4.056841419	0.034707495	0.045482393
100000	3	257.5684375	9.932281564	0.320571595	0.043385284
	10	627.944	2.752059592	0.233746217	0.122789626
	20	1236.7605	0.441941738	0.086816224	0.179101229

TABLA A.4: Tabla de resultados de la experimentación serializada del algoritmo BA en el entrenamiento para una función de dos dimensiones.

BA - 2 dimensiones - serializado					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	2.2891	0.147116318	0.004546749	0.001914855
	10	7.1623	0.210335209	0.0045498	0.002407068
	20	14.0734	0.179242356	0.001613773	0.001094625
10000	3	21.236375	0.185837054	2.290340088	0.007713888
	10	67.8255	0.545197059	2.210673869	0.365900114
	20	132.1409	0.253092188	0.688745819	0.398442283
100000	3	202.8481667	3.192399061	2.499579384	0.0000499
	10	663.9664	1.275324813	2.490430917	0.25868171
	20	1320.0909	8.650415372	1.706820241	0.225193069

TABLA A.5: Tabla de resultados de la experimentación serializada del algoritmo GWO en el entrenamiento para una función de una dimensión.

GWO - 1 dimensión					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	2.2882	0.226800353	0.001523533	0.000571979
	10	6.9692	0.467148513	0.00107316	0.000445286
	20	12.9994	0.094378905	0.00061567	0.000338948
10000	3	21.2442	0.148355279	0.682959372	0.011570003
	10	62.1089	0.080603901	2.302532371	0.576540334
	20	122.1787	1.091438505	1.37933944	0.298351103
100000	3	205.7414	3.866476582	11.58404486	0.000797136
	10	615.46	1.473730037	5.817928899	0.488466507
	20	1214.7291	2.470067451	0.706024136	0.470945751

TABLA A.6: Tabla de resultados de la experimentación serializada del algoritmo GWO en el entrenamiento para una función de dos dimensiones.

GWO - 2 dimensiones					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	2.5183	0.076867924	0.057390077	0.043263971
	10	7.7944	0.022046416	0.037627424	0.032306744
	20	15.39233333	0.056422218	0.018575212	0.010901984
10000	3	25.24322222	0.431729886	0.05722258	0.04252631
	10	76.39744444	0.704531602	0.032221666	0.014893364
	20	149.7227778	0.448250984	0.030749107	0.008538908
100000	3	241.925	2.350433985	0.060383458	0.031717427
	10	774.6928	14.41809774	0.03706374	0.015151588
	20	1534.918	9.869435792	0.026658901	0.010769453

TABLA A.7: Tabla de resultados de la experimentación serializada del algoritmo PSO en el entrenamiento para una función de una dimensión.

PSO - 1 dimensión					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	2.2955	0.126384994	0.001370499	0.000207492
	10	6.8845	0.339815718	0.001147634	0.000285587
	20	14.5711	2.163504023	0.000529885	0.000235364
10000	3	21.6124	0.24039144	0.66639457	0.011367133
	10	63.2173	0.483504579	2.258434318	0.40686513
	20	123.5194	0.166549425	2.233649941	0.196754588
100000	3	202.9407	4.079743076	11.7787105	0.000536675
	10	616.4001	7.405996136	6.117295657	0.278871616
	20	1201.1118	9.10729989	5.70603131	0.302971609

TABLA A.8: Tabla de resultados de la experimentación serializada del algoritmo PSO en el entrenamiento para una función de dos dimensiones.

PSO - 2 dimensiones					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	2.4501	0.014464132	0.064786279	0.037677462
	10	7.8556	0.104988042	0.019551346	0.010877785
	20	15.27944444	0.047600187	0.014101979	8.18406E-05
10000	3	25.0421	0.204717936	0.057598943	0.027576108
	10	76.5546	0.677318274	0.029971875	0.008572069
	20	158.1722	7.55273916	0.022716608	0.005270145
100000	3	244.774625	8.593156	0.088065353	0.044797828
	10	785.175	6.9712483	0.030116047	0.01723824
	20	1678.8957	7.9837484	0.02575437	0.009292395

TABLA A.9: Tabla de resultados de la experimentación serializada del algoritmo PFO en el entrenamiento para una función de una dimensión.

PFO - 1 dimensión					
Reg_Number	**Molecules**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
100000	3	3.0885	0.05783	0.098483	0.01360011
	10	7.29873	0.08305	0.087731	0.01126689
	20	12.0559	0.03481	0.036649	0.00476573
100000	3	34.9985	0.98615	0.12385	0.01864996
	10	63.4873	1.47394	0.100563	0.01385671
	20	112.5889	1.06589	0.087593	0.01062181
100000	3	220.7439	2.85949	0.298471	0.04852712
	10	603.7355	5.39438	0.2117047	0.04054138
	20	1197.4958	8.46973	0.158632	0.01652165

TABLA A.10: Tabla de resultados de la experimentación serializada del algoritmo PFO en el entrenamiento para una función de dos dimensiones.

PFO - 2 dimensiones					
Reg_Number	**Molecules**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	3.49885	0.05783	0.098483	0.01360011
	10	7.29873	0.0830511	0.087731	0.01126689
	20	14.27048	0.034801	0.036649	0.00476573
10000	3	34.9985	0.9861	0.12385	0.01864996
	10	77.04607	1.47394	0.100563	0.01385671
	20	140.98453	1.06589348	0.087593	0.01062181
100000	3	279.44381	2.85949834	0.298471	0.04852712
	10	775.54023	5.39438	0.2117047	0.04054138
	20	1482.75016	8.466973	0.158632	0.01652165

TABLA A.11: Tabla de resultados de la experimentación del algoritmo paralelizado BA en el entrenamiento para una función de una dimensión.

BA - 1 dimensión					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	0.862	0.057447753	0.28509714	0.033245079
	10	0.8997	0.057515306	0.305111994	0.068336354
	20	1.241	0.05283944	0.405783154	0.088747905
10000	3	2.551	0.067675233	0.241816497	0.07677685
	10	3.778	0.125192265	0.307954525	0.066059394
	20	5.0519	0.134969716	0.463394402	0.08627299
100000	3	23.5831	0.513529111	0.274444713	8.36E-03
	10	33.2037	0.59798878	0.366331064	0.059733434
	20	47.2817	0.656996712	0.442507229	0.153996571

TABLA A.12: Tabla de resultados de la experimentación del algoritmo paralelizado BA en el entrenamiento para una función de dos dimensiones.

BA - 2 dimensiones					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	0.8856	0.024281002	0.040079464	0.002823722
	10	1.0328	0.026328941	0.042605079	0.003068737
	20	1.7661	0.030549612	0.003340247	0.001201101
10000	3	2.8405	0.071929075	1.065196271	0.335242554
	10	3.7603	0.058088597	1.22128115	0.211865279
	20	5.8857	0.074998708	1.203226459	0.23720542
100000	3	24.9751	0.204972501	2.36229339	0.076323755
	10	32.5079	0.231762024	6.385393588	0.285704509
	20	46.2261	0.097526743	12.50311029	0.47102672

TABLA A.13: Tabla de resultados de la experimentación del algoritmo paralelizado GWO en el entrenamiento para una función de una dimensión.

GWO - 1 dimensión					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	0.75656	0.020081777	0.080873209	0.058419787
	10	1.0843	0.075119239	0.135967776	0.00627509
	20	1.1776	0.09698591	0.253309003	0.017361439
10000	3	2.821	0.270480642	0.357131769	0.053823254
	10	3.5487	0.058233343	0.214412942	0.048155995
	20	4.986	0.072363281	0.146430521	0.034808645
100000	3	22.17	0.175882537	0.368772208	0.040024565
	10	30.7832	0.110714648	0.197502873	0.035577742
	20	43.6299	0.471992338	0.127439109	0.056792303

TABLA A.14: Tabla de resultados de la experimentación del algoritmo paralelizado GWO en el entrenamiento para una función de dos dimensiones.

GWO - 2 dimensiones					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	0.79989	0.053938494	0.004047852	0.001027217
	10	0.9141	0.037173617	0.001261454	0.000642297
	20	1.0391	0.029968317	0.000575868	0.000276497
10000	3	2.6353	0.016773325	1.093524341	0.272404472
	10	3.6388	0.117110394	1.95669313	0.551905206
	20	5.08	0.050048865	1.99105274	0.564679718
100000	3	23.5803	0.443954214	10.79624147	0.515563165
	10	31.3566	0.246981421	5.591944217	0.508456637
	20	44.0297	0.225786551	1.562495959	0.883936299

TABLA A.15: Tabla de resultados de la experimentación del algoritmo paralelizado PSO en el entrenamiento para una función de una dimensión.

PSO - 1 dimensión					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	0.762	0.028620117	0.008089223	0.025539582
	10	0.8766	0.028558711	0.003775857	0.062043219
	20	1.0413	0.026208777	0.003161443	0.075337305
10000	3	2.551	0.033575785	1.044523871	0.061453741
	10	3.5605	0.062193694	1.217068431	0.053209364
	20	5.0207	0.067042358	1.165183154	0.071200136
100000	3	22.3518	0.254548315	11.63643509	8.37E-05
	10	31.5109	0.584745044	5.963666286	0.03044172
	20	45.1917	0.640528263	2.077968356	0.068453608

TABLA A.16: Tabla de resultados de la experimentación del algoritmo paralelizado PSO en el entrenamiento para una función de dos dimensiones.

PSO - 2 dimensiones					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	0.7956	0.02392674	0.259675247	0.002697733
	10	0.9028	0.024706275	0.291608016	0.00291671
	20	1.0377	0.028526985	0.394801699	0.001152469
10000	3	2.651	0.070457237	0.240152423	0.331190483
	10	3.6439	0.055060472	0.302966662	0.207031067
	20	5.1185	0.074497204	0.451213896	0.227905209
100000	3	23.5793	0.198828262	0.271992972	0.075667919
	10	31.4097	0.212409484	0.333382205	0.265135106
	20	44.3637	0.090169039	0.434069824	0.44001899

TABLA A.17: Tabla de resultados de la experimentación del algoritmo paralelizado PFO en el entrenamiento para una función de una dimensión.

PFO - 1 dimensión					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	2.804	0.250674202	0.156033093	0.047891655
	10	2.9499	0.031813519	0.222526576	0.074981464
	20	3.1601	0.088861253	0.232349659	0.050123256
10000	3	4.7543	0.1105522	0.271153587	0.044096722
	10	5.6634	0.110743146	0.231430859	0.055889276
	20	7.1491	0.188333894	0.123711698	0.038851
100000	3	20.259	0.357039058	0.283599713	0.060113236
	10	31.0652	0.192478455	0.245698138	0.034674257
	20	40.2095	0.493085354	0.171910199	0.02819845

TABLA A.18: Tabla de resultados de la experimentación del algoritmo paralelizado PFO en el entrenamiento para una función de dos dimensiones.

PFO - 2 dimensiones					
Reg_Number	**Moléculas**	**ET_Mean**	**ET_stDev**	**RMSE_Mean**	**RMSE_stDev**
1000	3	2.791222222	0.05870222	0.004096963	0.001534408
	10	2.9042	0.039832427	0.001662881	0.000867424
	20	3.0743	0.051259796	0.000654709	0.000246022
10000	3	4.69	0.032584932	0.704734882	0.000986252
	10	5.6793	0.091366235	0.621787054	0.000886816
	20	7.0932	0.045165867	0.615310319	0.001013887
100000	3	22.9779	0.301459948	0.996894646	0.005708911
	10	31.7068	0.704436702	0.721720694	0.006773227
	20	43.5416	0.158522484	0.706909573	0.003842571

Bibliografía

[1] Ali Akoglu y Gregory M Striemer. «Scalable and highly parallel implementation of Smith-Waterman on graphics processing unit using CUDA». En: *Cluster Computing* 12.3 (2009), págs. 341-352.

[2] Thomas Beielstein, Konstantinos E Parsopoulos y Michael N Vrahatis. *Tuning PSO parameters through sensitivity analysis*. Universitätsbibliothek Dortmund, 2002.

[3] Ron Bekkerman, Mikhail Bilenko y John Langford. *Scaling up machine learning*. Cambridge University Press, 2012, págs. 1-1. ISBN: 0521192242. DOI: 10.1145/2107736.2107740.

[4] Serge Thomas Mickala Bourobou y Younghwan Yoo. «User activity recognition in smart homes using pattern clustering applied to temporal ANN algorithm». En: *Sensors* 15.5 (2015), págs. 11953-11971.

[5] Lucélia Nobre Carvalho y col. «Feeding habits and habitat use of three sympatric piranha species in the Pantanal wetland of Brazil». En: *Ichthyological exploration of freshwaters* 18.2 (2007), pág. 109.

[6] KG Manosha Chathuramali y Ranga Rodrigo. «Faster human activity recognition with SVM». En: *International Conference on Advances in ICT for Emerging Regions (ICTer2012)*. IEEE. 2012, págs. 197-203.

[7] Nitesh V Chawla y col. «SMOTE: synthetic minority over-sampling technique». En: *Journal of artificial intelligence research* 16 (2002), págs. 321-357.

[8] Long Cheng y col. «Recognition of human activities using machine learning methods with wearable sensors». En: *2017 IEEE 7th annual computing and communication workshop and conference (CCWC)*. IEEE. 2017, págs. 1-7.

[9] Kalyanmoy Deb. «Multi-objective optimization». En: *Search methodologies*. Springer, 2014, págs. 403-449.

[10] Jean Demaison. «2 The Method of Least Squares». En: *Equilibrium Molecular Structures* (2011), págs. 1-7. DOI: 10.1201/b10374-3.

[11] Yongsheng Ding, Lei Chen y Kuangrong Hao. «Bio-Inspired Optimization Algorithms». En: 8.9 (2018), págs. 317-391. DOI: 10.1007/978-981-10-6689-4_8. URL: http://link.springer.com/10.1007/978-981-10-6689-4{_}8.

[12] Yu M Ermoliev y RJ-B Wets. *Numerical techniques for stochastic optimization*. Springer-Verlag, 1988.

[13] Lin Fan, Zhongmin Wang y Hai Wang. «Human activity recognition model based on decision tree». En: *2013 International Conference on Advanced Cloud and Big Data*. IEEE. 2013, págs. 64-68.

[14] Hossam Faris y col. «Grey wolf optimizer: a review of recent variants and applications». En: *Neural computing and applications* 30.2 (2018), págs. 413-435.

[15] Richard M. Foxx. «Attack preferences of the red-bellied piranha (Serrasalmus nattereri)». En: *Animal Behaviour* 20.2 (1972), págs. 280-283. ISSN: 0003-3472. DOI: 10.

1016/S0003-3472(72)80049-6. URL: https://www.sciencedirect.com/science/article/pii/S0003347272800496?via{\%}3Dihub.

[16] Michael J. Franklin y col. «Apache Spark». En: *Communications of the ACM* 59.11 (2016), págs. 56-65. ISSN: 00010782. DOI: 10.1145/2934664. URL: http://dl.acm.org/citation.cfm?doid=3013530.2934664.

[17] Amir Hossein Gandomi y col. «Bat algorithm for constrained optimization tasks». En: *Neural Computing and Applications* 22.6 (2013), págs. 1239-1255.

[18] V.G. Gudise y G.K. Venayagamoorthy. «Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks». En: *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*. Vol. 2. 1. IEEE, págs. 110-117. ISBN: 0-7803-7914-4. DOI: 10.1109/SIS.2003.1202255. URL: http://ieeexplore.ieee.org/document/1202255/.

[19] Daniel P Heyman y Matthew J Sobel. *Stochastic models in operations research: stochastic optimization*. Vol. 2. Courier Corporation, 2004.

[20] Jarmo Ilonen, Joni Kristian Kamarainen y Jouni Lampinen. «Differential evolution training algorithm for feed-forward neural networks». En: *Neural Processing Letters* 17.1 (2003), págs. 93-105. ISSN: 13704621. DOI: 10.1023/A:1022995128597.

[21] T Jayalakshmi y A Santhakumaran. «Statistical normalization and back propagation for classification». En: *International Journal of Computer Theory and Engineering* 3.1 (2011), págs. 1793-8201.

[22] Liheng Jian y col. «Parallel data mining techniques on graphics processing unit with compute unified device architecture (CUDA)». En: *The Journal of Supercomputing* 64.3 (2013), págs. 942-967.

[23] Morgan Kaufmann. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs (Applications of GPU Computing Series)*. Morgan Kaufmann, 2012, pág. 600. ISBN: 0124159885. URL: https://books.google.com.mx/books?id=EX2LNkSqViUC{\&}dq=cuda+parallel{\&}lr={\&}source=gbs{_}navlinks{_}s.

[24] Carl T Kelley. *Iterative methods for optimization*. SIAM, 1999.

[25] Siu Kwan Lam, Antoine Pitrou y Stanley Seibert. «Numba: A llvm-based python jit compiler». En: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, págs. 1-6.

[26] Oscar D. Lara y Miguel A. Labrador. «A Survey on Human Activity Recognition using Wearable Sensors». En: *IEEE Communications Surveys & Tutorials* 15.3 (2013), págs. 1192-1209. ISSN: 1553-877X. DOI: 10.1109/SURV.2012.110112.00192. URL: http://ieeexplore.ieee.org/document/6365160/.

[27] Ali Madadi y Mahmood Mohseni Motlagh. «Optimal Control of DC motor using Grey Wolf Optimizer Algorithm». En: *Technical Journal of Engineering and Applied Sciences* 4 (2014), págs. 373-379. ISSN: 2051-0853.

[28] Andrea Mannini y Angelo Maria Sabatini. «Machine learning methods for classifying human physical activity from on-body accelerometers». En: *Sensors* 10.2 (2010), págs. 1154-1175.

[29] Lourdes Martínez-Villaseñor y col. «UP-fall detection dataset: A multimodal approach». En: *Sensors* 19.9 (2019), pág. 1988.

[30] Xiangrui Meng y col. *MLlib: Machine Learning in Apache Spark*. Inf. téc. 2015, págs. 1-7. arXiv: 1505.06807. URL: http://arxiv.org/abs/1505.06807.

[31] Sandie Millot, Pierre Vandewalle y Eric Parmentier. «Sound production in red-bellied piranhas (Pygocentrus nattereri, Kner): an acoustical, behavioural and morphofunctional study.» En: *The Journal of experimental biology* 214.Pt 21 (2011), págs. 3613-8. ISSN: 1477-9145. DOI: 10.1242/jeb.061218. URL: http://www.ncbi.nlm.nih.gov/pubmed/21993790.

[32] Seyedali Mirjalili. «How effective is the Grey Wolf optimizer in training multi-layer perceptrons». En: *Applied Intelligence* 43.1 (2015), págs. 150-161. ISSN: 0924-669X. DOI: 10.1007/s10489-014-0645-7. URL: http://ieeexplore.ieee.org/document/329294/http://link.springer.com/10.1007/s10489-014-0645-7.

[33] Seyedali Mirjalili, Seyed Mohammad Mirjalili y Andrew Lewis. «Grey Wolf Optimizer». En: *Advances in Engineering Software* 69 (2014), págs. 46-61. ISSN: 09659978. DOI: 10.1016/j.advengsoft.2013.12.007. URL: http://dx.doi.org/10.1016/j.advengsoft.2013.12.007.

[34] Seyedali Mirjalili y col. «Multi-objective grey wolf optimizer: A novel algorithm for multi-criterion optimization». En: *Expert Systems with Applications* 47 (2016), págs. 106-119. ISSN: 09574174. DOI: 10.1016/j.eswa.2015.10.039. URL: http://dx.doi.org/10.1016/j.eswa.2015.10.039.

[35] Volodymyr Mnih. *Cudamat: a CUDA-based matrix class for python*. Inf. téc. 2009. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.232.4776{\&}rep=rep1{\&}type=pdf.

[36] Ferenc Molnar Jr y col. «Air pollution modelling using a Graphics Processing Unit with CUDA». En: *Computer Physics Communications* 181.1 (2010), págs. 105-112.

[37] David J Montana y Lawrence Davis. «Training feedforward neural networks using genetic algorithms». En: *Proceedings of the International Joint Conference on Artificial Intelligence* (1989), págs. 762-767. ISSN: 0019-1604 (Print) 0019-1604 (Linking). DOI: 10.1016/j.gerinurse.2005.08.002. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.3838{\&}rep=rep1{\&}type=pdf{\%}5Cnpapers2://publication/uuid/40B24425-E482-4CDD-A215-0D673E02463E.

[38] Satyasai Jagannath Nanda y Ganapati Panda. «A survey on nature inspired metaheuristic algorithms for partitional clustering». En: *Swarm and Evolutionary Computation* 16 (2014), págs. 1-18. ISSN: 22106502. DOI: 10.1016/j.swevo.2013.11.003. URL: http://dx.doi.org/10.1016/j.swevo.2013.11.003.

[39] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[40] Hiram Ponce, Luis Miralles-Pechuán y María De Lourdes Martínez-Villaseñor. «A flexible approach for human activity recognition using artificial hydrocarbon networks». En: *Sensors (Switzerland)* 16.11 (2016), págs. 1-24. ISSN: 14248220. DOI: 10.3390/s16111715.

[41] Hiram Ponce y Pedro Ponce. «Método de Aprendizaje Automático Basado en Compuestos Orgánicos». En: III (2007).

[42] Hiram Ponce-Espinosa, Pedro Ponce-Cruz y Arturo Molina. «Artificial Hydrocarbon Networks». En: 2014, págs. 73-111. DOI: 10.1007/978-3-319-02472-1_4. URL: http://link.springer.com/10.1007/978-3-319-02472-1{_}4.

[43] Hiram Ponce-Espinosa, Pedro Ponce-Cruz y Arturo Molina. «Artificial Organic Networks». En: (2014), págs. 53-72. DOI: 10.1007/978-3-319-02472-1_3. URL: http://link.springer.com/10.1007/978-3-319-02472-1{_}3.

[44] Kedar Potdar, Taher S Pardawala y Chinmay D Pai. «A comparative study of categorical variable encoding techniques for neural network classifiers». En: *International journal of computer applications* 175.4 (2017), págs. 7-9.

[45] Helder Queiroz y Anne E Magurran. «Safety in numbers? Shoaling behaviour of the Amazonian red-bellied piranha». En: *Biology Letters* 1.2 (2005), págs. 155-157. ISSN: 1744-9561. DOI: 10.1098/rsbl.2004.0267. URL: http://www.royalsocietypublishing.org/doi/10.1098/rsbl.2004.0267.

[46] Andrés Ramos y col. «Modelos matemáticos de optimización». En: *Publicación Técnica* 1 (2010).

[47] Jason Sanders y Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[48] Ivan Sazima y Francisco A. Machado. «Underwater observations of piranhas in western Brazil». En: Springer, Dordrecht, 1990, págs. 17-31. DOI: 10.1007/978-94-009-2065-1_2. URL: http://www.springerlink.com/index/10.1007/978-94-009-2065-1{_}2.

[49] James G. Shanahan y Laing Dai. «Large Scale Distributed Data Science using Apache Spark». En: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*. New York, New York, USA: ACM Press, 2015, págs. 2323-2324. ISBN: 9781450336642. DOI: 10.1145/2783258.2789993. URL: http://dl.acm.org/citation.cfm?doid=2783258.2789993.

[50] Nazmul Siddique y Hojjat Adeli. «Nature Inspired Computing: An Overview and Some Future Directions». En: *Cognitive Computation* 7.6 (2015), págs. 706-714. ISSN: 18669964. DOI: 10.1007/s12559-015-9370-8.

[51] Deepika Singh y col. «Human activity recognition using recurrent neural networks». En: *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer. 2017, págs. 267-274.

[52] Slawomir Koziel and Xin-She Yang. *Computational Optimization, Methods and Algorithms*. Ed. por Slawomir Koziel y Xin-She Yang. Vol. 356. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. ISBN: 978-3-642-20858-4. DOI: 10.1007/978-3-642-20859-1. URL: http://link.springer.com/10.1007/978-3-642-20859-1.

[53] Jan A Snyman y Daniel N Wilke. *Practical mathematical optimization: basic optimization theory and gradient-based algorithms*. Vol. 133. Springer, 2018.

[54] Mohammad S Sorower. «A literature survey on algorithms for multi-label learning». En: ().

[55] Anton Stabentheiner. «Correlations between hearing and sound production in piranhas». En: *Journal of Comparative Physiology A* 162.1 (1988), págs. 67-76. ISSN: 0340-7594. DOI: 10.1007/BF01342704. URL: http://link.springer.com/10.1007/BF01342704.

[56] AS Abdull Sukor, A Zakaria y N Abdul Rahim. «Activity recognition using accelerometer sensor and machine learning classifiers». En: *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*. IEEE. 2018, págs. 233-238.

[57] Caglar Tirkaz y col. «Activity recognition using a hierarchical model». En: *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*. IEEE. 2012, págs. 2814-2820.

[58] Pavan Turaga y col. «Machine recognition of human activities: A survey». En: *IEEE Transactions on Circuits and Systems for Video technology* 18.11 (2008), págs. 1473-1488.

[59] Peter JM Van Laarhoven y Emile HL Aarts. «Simulated annealing». En: *Simulated annealing: Theory and applications*. Springer, 1987, págs. 7-15.

[60] Grega Vrbančič y col. «NiaPy: Python microframework for building nature-inspired algorithms». En: *Journal of Open Source Software* 3.23 (2018), pág. 613. ISSN: 2475-9066. DOI: 10.21105/joss.00613. URL: http://joss.theoj.org/papers/10.21105/joss.00613.

[61] Lei Wang y col. «Task scheduling of parallel processing in CPU-GPU collaborative environment». En: *2008 International Conference on Computer Science and Information Technology*. IEEE. 2008, págs. 228-232.

[62] D.H. Wolpert y W.G. Macready. «No free lunch theorems for optimization». En: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), págs. 67-82. ISSN: 1089778X. DOI: 10.1109/4235.585893. URL: http://ieeexplore.ieee.org/document/585893/.

[63] Fei Xue y col. «Optimal parameter settings for bat algorithm». En: *International Journal of Bio-Inspired Computation* 7.2 (2015), págs. 125-128.

[64] Xin-She Yang. «A {New} {Metaheuristic} {Bat}-{Inspired} {Algorithm}». En: *Nature {Inspired} {Cooperative} {Strategies} for {Optimization} ({NICSO} 2010)* 284 (2010), págs. 65-74. DOI: 10.1007/978-3-642-12538-6_6. URL: http://link.springer.com/10.1007/978-3-642-12538-6{_}6.

[65] Xin-She Yang y Amir Hossein Gandomi. «Bat algorithm: a novel approach for global engineering optimization». En: *Engineering computations* (2012).

[66] Xin-She Yang y Xingshi He. «Bat algorithm: literature review and applications». En: *International Journal of Bio-inspired computation* 5.3 (2013), págs. 141-149.

[67] Xizhe Yin y col. «Human activity detection based on multiple smart phone sensors and machine learning algorithms». En: *2015 IEEE 19th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE. 2015, págs. 582-587.