Algoritmo cuántico para resolver el problema de la mochila binaria en instancias de baja dimensionalidad

DANILO LÓPEZ SANDOVAL

Director: PhD. CARLOS ALBERTO COBOS LOZADA

UNIVERSIDAD DEL CAUCA

FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

DEPARTAMENTO DE SISTEMAS

GRUPO DE I+D EN TECNOLOGÍAS DE LA INFORMACIÓN (GTI)

LÍNEA INVESTIGACIÓN EN SISTEMAS INTELIGENTES

POPAYÁN, MARZO DE 2022

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

Anexo A. Código fuente de la aplicación desarrollada en Python.

Anexo B. Artículo publicado, titulado: *Computación Cuántica Adiabática aplicada a la solución del Problema de la Mochila Binaria.*

Anexo C. Artículo final para iniciar proceso de publicación, titulado: *Análisis comparativo de eficiencia del módulo de optimización de Qiskit para dar solución al Problema de la Mochila Binaria.*

Anexo D. Diagramas de clase especifico de la aplicación.

Anexo E. Archivos de resultados y análisis de pruebas realizadas.

RESUMEN

Día a día los problemas que enfrenta la humanidad son más complejos, y las herramientas que nos brinda la computación clásica empiezan a ser insuficientes para resolverlos, como por ejemplo los problemas de optimización donde un espacio de búsqueda puede crecer exponencialmente; es así como los avances recientes en la investigación cuántica han llevado al desarrollo de algoritmos cuánticos experimentales que prometen resolver dichos problemas de optimización y han demostrado su superioridad ante algunos de sus análogos clásicos. En este contexto, la computación cuántica ha empezado a mostrar su dominio y en los últimos años empresas como IBM o Microsoft empiezan a ofertar servicios de cómputo cuántico para tareas de optimización y análisis de datos.

Con este nuevo paradigma se podrán estudiar problemas de alta complejidad que tienen gran cantidad de operaciones y manejan gran cantidad de variables, para esto, se hace uso de algunas propiedades de la física cuántica como el entrelazamiento cuántico o la superposición cuántica, con las cuales se pueden realizar más operaciones en una misma unidad de tiempo disminuyendo radicalmente los tiempos de respuesta. Debido a que la aplicabilidad de dichos algoritmos sobre un dispositivo cuántico para muchos problemas de optimización teóricos y del mundo real, que a menudo contienen gran cantidad de restricciones, se encuentra limitada por el número actual de qubits que se han logrado integrar en un sistema físico estable, en los últimos años se han realizado estudios para tratar de aplicar algunas leyes del campo cuántico a algoritmos que funcionan sobre un sistema de computación clásico.

Un problema de optimización particularmente conocido y estudiado es el problema de la mochila binaria, el cual sirve de base para dar solución a una gran variedad de problemas combinatorios complejos. Ya que se han venido desarrollando soluciones de algoritmos híbridos que incluyen componentes cuánticos, se hace necesario comprender los conceptos claves de la computación cuántica, los modelos hamiltonianos, la computación cuántica adiabática y el desarrollo actual de soluciones cuánticas que permitan obtener resultados comparables a las soluciones convencionales.

Así, en este documento se detallan los pasos seguidos para construir una solución al problema de la mochila binaria utilizando un algoritmo de Computación Cuántica Adiabática haciendo uso de las librerías de Qiskit y su implementación en Python. Para validar la eficiencia del algoritmo propuesto, se realizaron experimentos con un conjunto de instancias de diversa complejidad y correlación, y se comparan los resultados de la propuesta cuántica con tres algoritmos del estado del arte.

Los resultados arrojados indican que el algoritmo cuántico se encuentra al nivel del algoritmo clásico que obtuvo los mejores resultados y es más eficiente y eficaz que dos propuestas de algoritmos evolutivos. Se concluye discutiendo la viabilidad experimental del algoritmo cuántico evaluado, así como su potencial para encontrar soluciones viables y reducir los requisitos de recursos para implementar soluciones a estos problemas de optimización que permitan dirigir la búsqueda del conocimiento necesario para lograr avances en el campo de la implementación de algoritmos que integren conocimientos del área de computación cuántica y brinden un rendimiento superior a los algoritmos clásicos actuales.

Esta página ha sido dejada intencionalmente en blanco.

CAPÍTULO 1

1 INTRODUCCIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

El problema de la mochila es un conocido problema de optimización combinatoria que pertenece a la clase de problemas NP-completos [1]. Existen diferentes variantes del problema de la mochila que se utilizan como guía para resolver una gran variedad de problemas, entre ellos, problemas de empaque y reducción de existencias, toma de decisiones financieras, titulización respaldada por activos, subastas combinatorias [2], toma de decisiones y corte de material [3].

Entre las variantes más conocidas del problema de la mochila se encuentran: la mochila binaria (Binary Knapsack Problem, BKP), el problema de la suma de subconjuntos ($P_i = W_i$), el problema de la mochila sin límite, el problema de la mochila acotada [4], el problema de la mochila d-dimensional (d-KP), el problema de las múltiples mochilas (MKP) [5], el problema de la mochila de elección múltiple multidimensional y el problema de la mochila cuadrática [6]. A la fecha se han desarrollado una variedad de técnicas para resolver las diferentes variantes del problema de la mochila, las cuales se pueden agrupar de la siguiente manera: (i) exactos, (ii) programación dinámica, (iii) programación entera, (iv) métodos metaheurísticos, (v) métodos lagrangianos, (vi) métodos basados en árboles de búsqueda con back tracking y (vii) enfoques de red [7].

Esta investigación se relaciona con el problema de la mochila binaria que se define formalmente por la **Ecuación (1)** y se describe de la siguiente forma: dada una mochila con una capacidad limitada **C** $\in Z^+$, y un conjunto de ***n*** elementos (artículos o ítems), cada uno con un beneficio $\mathbf{P_i} \in Z^+$ y un peso $\mathbf{W_i} \in Z^+$ donde i = 1, 2, ..., n, se debe seleccionar un subconjunto de ***m*** elementos ($m \leq n$) de modo que se genere la mayor (máxima) ganancia o beneficio posible, sujeto a una restricción principal la cual define que los pesos totales de los elementos seleccionados no excedan la capacidad **C** de la mochila [3], teniendo en cuenta que x_i es un valor binario {0,1} que indica si el elemento *i* debe ir o no en la mochila.

$$Max\ f(x) = \sum_{i=1}^{n} P_i \times x_i$$
$$sujeto\ a \sum_{i=1}^{n} W_i \times x_i \leq C \tag{1}$$

$$x_i \in \{0,1\}, i = 1, 2, \dots, n$$

Existen diferentes maneras de clasificar un problema de mochila binaria, entre las más destacadas se tiene una definición basada en el número de elementos n (baja o pequeña dimensionalidad si n < 20, dimensionalidad media si 20 ≤ n < 200 y alta dimensionalidad si n ≥ 1000 [8]) y una definición mejor aceptada de clasificación de acuerdo con la complejidad de las instancias, esto es la relación o no de los pesos de cada ítem con su valor, y se definen como instancias no correlacionadas, débilmente correlacionadas, casi fuertemente correlacionadas, fuertemente correlacionadas, inversamente correlacionadas, e instancias con pesos y beneficios iguales (subconjuntos de instancias de suma) [9].

Debido a la importancia y el reto que representa el problema de la mochila binaria, en los últimos años se han reportado un gran número de algoritmos que buscan su solución, estos se agrupan en: exactos, de programación dinámica, basados en back-tracking (incluidos ramificación y poda), y metaheurísticos. Entre los algoritmos metaheurísticos más destacados se encuentran los algoritmos genéticos, el recocido simulado, la optimización por enjambre de partículas (PSO) y la búsqueda tabú [3], [10]. Además, recientemente se han realizado investigaciones con algoritmos inspirados en conceptos de la física cuántica como el algoritmo evolucionario cuántico [11], el algoritmo genético cuántico [12] y el algoritmo VQE (Variational Quantum Eigensolver) [13], con lo cual empieza a vislumbrarse una línea de investigación en este ámbito.

Por su parte, la computación cuántica nace como una alternativa al paradigma computacional convencional basado en máquinas de Turing y de Von Neumann, la cual ha demostrado su superioridad ante la computación clásica para algunos problemas específicos [14], [15]. Con este nuevo paradigma se pueden estudiar problemas de alta complejidad que tienen gran cantidad de operaciones y manejan gran cantidad de variables, para esto, se hace uso de algunas propiedades de la física cuántica como el entrelazamiento cuántico o la superposición cuántica, con las cuales se pueden realizar más operaciones en una misma unidad de tiempo disminuyendo radicalmente los tiempos de respuesta [16].

A la fecha se han definido varios algoritmos cuánticos eficientes para problemas discretos como la factorización entera, la simulación cuántica, la estimación del valor propio, la integración, la solución de ecuaciones diferenciales parciales y la solución a problemas numéricos de álgebra lineal [17], pero una búsqueda en la literatura reveló que existen muy pocos artículos publicados sobre algoritmos cuánticos que se apliquen al problema de la mochila binaria [2][11].

Para lograr la solución de problemas combinatorios complejos como el de la mochila binaria, se hace necesario comprender los conceptos claves de la computación

cuántica, los modelos hamiltonianos, la computación cuántica adiabática y el desarrollo actual de soluciones a problemas concretos de optimización. Finalmente, realizar nuevas propuestas y comparar las soluciones obtenidas con los modelos cuánticos y sus contrapartes tradicionales para establecer si se obtienen mejoras. Iniciando este trabajo, el autor del realizó una primera revisión e implementación de un algoritmo cuántico para la solución del problema de la mochila binaria, encontrando que lo existente no permite la solución del 100% de las instancias evaluadas, ya que los modelos de Ising existentes tienden a llenar la mochila totalmente y en algunas instancias la solución óptima se encuentra dejando la mochila con espacio libre [18].

En este sentido, en este trabajo de grado se planteó la siguiente pregunta de investigación: ¿Cuáles son las características de un hamiltoniano cuántico basado en un modelo de Ising y su implementación, que al ser ejecutada en emuladores de computación cuántica permite obtener resultados comparables o mejores que los obtenidos mediante algoritmos clásicos del problema de la mochila binaria en instancias de baja dimensionalidad ($n < 20$)?

Es preciso aclarar dos aspectos de esta pregunta de investigación. Primero, se plantea el uso de emuladores de computación cuántica debido a que a la fecha no se cuenta con el acceso a computadores cuánticos que se programen con lenguajes de alto nivel como Python; existe la posibilidad de contar con tiempo limitado de procesamiento en un computador cuántico de IBM, pero programado a nivel de circuitos y compuertas cuánticas, lo que desborda el alcance y el objetivo central de la investigación. Y segundo, se plantea el uso de instancias de baja dimensionalidad ($n < 20$), esto debido a que los emuladores tienen restricciones para procesar problemas con mayor número de dimensiones.

1.2 APORTES DEL PROYECTO

Con el desarrollo de este proyecto se logró contribuir desde la perspectiva de la generación de nuevo conocimiento en la línea de investigación de Sistemas Inteligentes del Grupo de I+D en Tecnologías de la Información (GTI), mostrando una primer solución a un problema clásico de optimización con el enfoque de computación cuántica adiabática, dando a conocer el funcionamiento de un algoritmo adiabático cuántico aplicado al problema de la mochila binaria, y comparando los resultados obtenidos frente a algoritmos del estado del arte en instancias de baja dimensionalidad ($n < 20$).

Desde la perspectiva del desarrollo se contribuye con una framework en Python para evaluar algoritmos que resuelven el problema de la mochila binaria equipado con diversos problemas y la implementación de 3 metaheurísticos del estado del arte y la propuesta cuántica realizada. Este framework es de código abierto y queda disponible a la comunidad académica y científica para enriquecerlo y usarlo.

Se espera que las herramientas ofrecidas para solucionar problemas de optimización mediante computación cuántica evolucionen y que con este proyecto

se sienten las bases para que en el futuro se puedan realizar propuestas e investigaciones que manejen problemas de la mochila con alta dimensionalidad y se motive a los estudiantes a trabajar en esta interesante área de investigación.

1.3 OBJETIVOS

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

1.3.1 Objetivo general

Proponer un algoritmo cuántico para resolver instancias de baja dimensionalidad del problema de la mochila binaria basado en conceptos de computación cuántica adiabática.

1.3.2 Objetivos específicos

- Establecer la línea base de la investigación con el modelado e implementación de un marco de prueba implementado en Python que incluya tres (3) algoritmos del estado del arte que resuelven el problema de la mochila binaria en instancias de baja dimensionalidad ($n<20$) con diferentes grados de complejidad y métricas de comparación (tasa de éxito para encontrar el resultado óptimo, mejor óptimo promedio encontrado y tiempo de ejecución) reconocidas por la comunidad científica.
- Definir un algoritmo cuántico y su implementación en Python sobre Qiskit™ Aqua, usando el proceso de investigación iterativo propuesto por Pratt, para resolver el problema de la mochila binaria en instancias de baja dimensionalidad.
- Documentar el desempeño del del algoritmo cuántico propuesto, a partir de un estudio comparativo de los resultados de este respecto a los algoritmos del estado del arte implementados en el marco de prueba, en función de las métricas e instancias disponibles.

1.4 RESULTADOS OBTENIDOS

A continuación, se resumen los principales resultados obtenidos en la elaboración del presente trabajo de grado:

- **Monografía de trabajo de grado**: Se refiere al presente documento en el cual se presenta una introducción al trabajo (problema, aportes, objetivos y resultados obtenidos), el estado del arte en el campo de la optimización, especialmente en el problema de la mochila binaria, un framework de pruebas implementado con el cual se evalúan cuatro algoritmos con enfoques de solución diferentes y la definición, implementación y resultados de la comparación de los algoritmos implementados en el framework, por último, se presentan las

conclusiones y el trabajo futuro que el grupo de investigación puede realizar en el área de la optimización cuántica.

- **Framework de pruebas:** Se refiere al código fuente del marco de trabajo de pruebas construido para realizar los experimentos (coloquialmente pruebas) e implementar los algoritmos que dan solución a problemas de optimización de la mochila binaria. El código fuente del marco de trabajo se encuentra disponible en [19] y en un anexo digital (**Anexo A**) de esta monografía.
- **Ponencia:** el artículo titulado *"Computación Cuántica Adiabática aplicada a la solución del Problema de la Mochila Binaria"*, se presentó en las Jornadas iberoamericanas de ingeniería de software e ingeniería del Conocimiento, JIISIC'2020.
- **Artículo publicado:** en el marco de las JIISIC'2020, el comité de evaluación decidió que las memorias del artículo titulado *"Computación Cuántica Adiabática aplicada a la solución del Problema de la Mochila Binaria"* se publicaran en la Revista Ibérica de Sistemas e Tecnologias de Informação - RISTI (ISSN: 1646-9895). El **Anexo B** hace referencia al artículo publicado en el cual se muestra una solución cuántica a problemas de baja dimensionalidad con resultados entre aceptables y prometedores.
- **Artículo final de la investigación**: artículo con la descripción y resultados del marco de trabajo propuesto en este documento, el cual ha sido titulado "*Análisis comparativo de eficiencia del módulo de optimización de Qiskit" para dar solución al Problema de la Mochila Binaria* y que se encuentra en proceso de envío a una revista o evento nacional o internacional (Ver **Anexo C)**.

1.5 ESTRUCTURA DE LA MONOGRAFÍA

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

CAPITULO 1: INTRODUCCIÓN: Hace referencia al presente capitulo que introduce el tema de investigación, presenta la pregunta de investigación que originó el trabajo, los aportes al problema, también los objetivos (general y específicos) definidos en el anteproyecto, un breve resumen de los resultados obtenidos y finalmente la organización de la monografía.

CAPITULO 2: CONTEXTO TEÓRICO Y ESTADO DEL ARTE: En este capítulo se presentan conceptos teóricos relacionados con computación cuántica. Además, se presentan propuestas relevantes del estado del arte que se han venido aplicando y mejorando con el paso del tiempo para la resolución de problemas de optimización.

CAPITULO 3: MARCO DE TRABAJO PROPUESTO: En este capítulo se explica la estructura y funcionamiento general del marco de trabajo desarrollado para la ejecución de los experimentos, así como un ejemplo de construcción y ejecución de un experimento.

CAPITULO 4. PROPUESTA CUÁNTICA: Se presentan dos algoritmos para dar solución al problema de la mochila mediante lógica cuantía y se detallan los componentes necesarios para abordar un problema de optimización desde la perspectiva de computación cuántica utilizando la librería de Qiskit

CAPITULO 5: EXPERIMENTOS Y RESULTADOS: Se presenta el protocolo de experimentación y se presentan los resultados de los algoritmos de la línea base y del algoritmo propuesto en los distintos escenarios de prueba. También se presenta un análisis comparativo de los resultados soportado con las pruebas estadísticas no paramétricas de Friedman y Wilcoxon.

CAPITULO 6: CONCLUSIONES Y TRABAJOS FUTUROS: En este capítulo se presentan las conclusiones obtenidas al finalizar el trabajo de grado e ideas que el grupo de investigación espera realizar en un futuro cercano.

CAPITULO 7: BIBLIOGRAFIA: Este último capítulo contiene las referencias bibliográficas de sitios web, artículos y libros consultados para la realización del proyecto.

CAPÍTULO 2

2 CONTEXTO TEÓRICO Y ESTADO DEL ARTE

2.1 CONTEXTO TEÓRICO

2.1.1 Computación cuántica

La computación cuántica es un nuevo paradigma de computación que surgió como resultado de la fusión de la informática y la mecánica cuántica. El origen de la computación cuántica se remonta a principios de los 80's cuando Richard Feynman observó que algunos efectos de la mecánica cuántica no se pueden simular de manera eficiente en una computadora clásica [16].

En computación cuántica, un bit cuántico (Quantum bit, qubit) es la unidad de información más pequeña almacenada en una computadora cuántica de dos estados [20]. Al contrario del bit clásico el cual tiene dos valores posibles, "0" o "1", un qubit puede estar en el estado "1", en el estado "0" o en cualquier superposición de los dos estados. El estado de un qubit se puede representar mediante la notación de corchetes (Bra-Ket) presentada en la **Ecuación (2)**.

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2}$$

Donde $|\Psi\rangle$ denota más de un vector $\vec{\Psi}$ en algún espacio vectorial y representa una superposición lineal de la partícula dados los vectores de estado cuántico individuales. $|0\rangle\ y\ |1\rangle$ representan los valores de bit clásicos 0 y 1 respectivamente. α y β son números complejos tales que $|\alpha|^2 + |\beta|^2 = 1$, los cuales especifican las amplitudes de probabilidad de los estados correspondientes. Cuando se realiza la medición del estado de un qubit, se puede obtener cero con una probabilidad $|\alpha|^2$ o uno con una probabilidad $|\beta|^2$. Existe una particularidad en el ámbito cuántico, al observar el estado cuántico de un qubit, este colapsa a un solo estado, cero (0) o uno (1) [21].

Un sistema de n-qubits puede representar 2^n estados al mismo tiempo. Este crecimiento exponencial del espacio de estado con el número de qubits es lo que sugiere una aceleración exponencial de la computación en las computadoras cuánticas sobre las computadoras tradicionales.

Con este nuevo paradigma se pueden estudiar problemas de alta complejidad que tienen gran cantidad de operaciones y manejan gran cantidad de variables. Para esto, se hace uso de algunas propiedades de la física cuántica como el entrelazamiento cuántico o la superposición cuántica, con las cuales se pueden realizar más operaciones en una misma unidad de tiempo disminuyendo

radicalmente los tiempos de respuesta [16]. Entre los algoritmos cuánticos más famosos se encuentran el algoritmo de Shore's [22] utilizado para factorización numérica y el algoritmo de Grover's [23] utilizado para búsquedas en una base de datos no ordenada. Ambos algoritmos redujeron la complejidad de la solución al problema [24]; al igual que los algoritmos de cuánticos aplicados a la estimación del valor propio, la integración, la solución de Ecuaciones diferenciales parciales y la solución a problemas numéricos de álgebra lineal [17].

Durante la última década, la computación cuántica ha atraído un interés generalizado y ha inducido intensivas investigaciones, debido especialmente a su paralelismo innato que reduce la complejidad algorítmica. Tal capacidad de procesamiento en paralelo se puede utilizar para resolver problemas de optimización combinatoria que requieren la exploración de grandes espacios de posibles soluciones [25]. Debido a la complejidad de diseñar y probar algoritmos cuánticos complejos en una maquina real, algunos investigadores han optado por emular algunas propiedades de la computación cuántica en algoritmos tradicionales [25].

2.1.2 Computación cuántica adiabática (AQC)

La computación cuántica adiabática (Adiabatic Quantum Computation, AQC) es un enfoque equivalente al modelo de circuito de computación cuántica, adecuado para problemas del tipo de optimización combinatoria, incluyendo particiones, coberturas, particionado de árboles y gráficos, y satisfacción booleana [2].

Debido a sus inicios, la AQC puede considerarse como una clase particular de Recocido cuántico (Quantum Annealing, QA), la cual utiliza los principios de la mecánica cuántica para resolver problemas de optimización [15]. Desde la propuesta inicial de QA, ha habido mucho interés en la búsqueda de problemas prácticos donde pueda ser ventajoso con respecto a los algoritmos clásicos, particularmente el recocido simulado (Simulating annealing, SA). Muchos de estos enfoques transforman un problema computacional en un problema donde se debe encontrar el estado fundamental de un modelo Ising Spin Glass (ISG) cuántico, el cual, en el peor de los casos es un problema NP-completo [26].

El modelo Ising (una clase conveniente, restringida y ciertamente no universal de Hamiltoniano) tiene la versatilidad de codificar eficientemente muchos problemas NP y ha motivado la realización física de QA. En general las computadoras cuánticas universales no pueden resolver problemas NP-hard de manera eficiente, pero se ha encontrado evidencia en los sistemas experimentales de Ising cuántico que sugiere una aceleración cuántica sobre la computación tradicional debido al efecto del túnel cuántico (hace referencia a pasar de un estado A a un estado B no contiguo para evitar estancarse en óptimos locales) [26].

2.1.3 Hamiltoniano en computación cuántica adiabática

En un modelo de circuito de computación cuántica, un cálculo puede evolucionar en todo el espacio de Hilbert (es una generalización del espacio euclidiano, es un espacio de producto interior que es completo con respecto a la norma vectorial definida por el producto interior) y está codificado en una serie de puertas de lógica cuántica unitarias [15]. Por otro lado, en un modelo AQC el cálculo se realiza mediante un Hamiltoniano inicial (H_0), cuyo estado fundamental codifica la solución a un problema de interés, y otro hamiltoniano (H_P), cuyo estado fundamental es trivial. Entonces, si se prepara un sistema cuántico para estar en el estado fundamental H_0, y luego se cambia adiabáticamente el hamiltoniano por un tiempo T, de acuerdo con la **Ecuación (3)**, y T es lo suficientemente grande, al final, el estado cuántico en T devolverá una solución al problema de interés [15][26][27].

$$H(t) = (1 - s)H_0 + sH_p \tag{3}$$

Donde s $\in$ [0, 1], es un parámetro de tiempo considerado como una función dependiente del tiempo s(t) = $\frac{t}{T}$ para un tiempo total de evolución T (en general se puede considerar cualquier función que satisfaga s(0) = 0 y s(1) = 1), y H_0 y H_p no conmutan (Dos operadores no conmutan cuando se cumple que [A, B] = AB - BA $\neq$ 0 [28]). Con esta definición y debido al teorema adiabático de la mecánica cuántica el sistema cuántico permanecerá en el estado fundamental todo el tiempo.

2.1.4 Modelo Ising

Uno de los modelos más utilizados en física se llama el modelo Ising. Propuesto entre 1920 y 1930 por Ernst Ising y Wilhelm Lenz como una forma de entender el funcionamiento de los materiales magnéticos. El enfoque modela un material magnético como una colección de moléculas, cada una de las cuales tiene un espín que puede alinearse o anti-alinearse con un campo magnético aplicado h_i, y que interactúan entre sí con base en un campo de interacción J_{ij} [29].

En la **Ecuación (4)** se representa el modelo clásico de Ising, el cual se puede escribir como una función cuadrática de un conjunto de *n* giros, donde $s_i \in \{-1, +1\}$ representa el spin de la i-ésima partícula.

$$H(s_1, \dots, s_N) = -\sum_{i=1}^{n} J_{ij} s_i s_j - \sum_{i=1}^{n} h_i s_i \tag{4}$$

La **Ecuación (5)** representa un Hamiltoniano (H_p) como la versión cuántica del modelo Ising, donde s_i se reemplaza por σ_i^z en la **Ecuación (4)**.

$$H_p = H\left(\sigma_1^z, \ldots, \sigma_N^z\right) = -\sum_{i<j} J_{ij}\sigma_i^z\sigma_j^z - \sum_{i=1}^{n} h_i\sigma_i^z \quad (5)$$

$\sigma_i^z = \mathbb{I}^{\otimes(i-1)} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \otimes \mathbb{I}^{\otimes(n-i)}$ es una matriz de Pauli[1] que actúa sobre el i-ésimo spin en un espacio de Hilbert de N qubits $\{|+\rangle, |-\}^{\otimes N}$ donde $\mathbb{I}$ es una matriz identidad de 2 x 2 y h_i , $J_{ij} \in \mathbb{R}$ son coeficientes [27][26]. h_i es la fuerza del campo aplicado sobre el i-ésimo spin y J_{ij} actúa como el campo de interacción entre los spines vecinos i, j [30]. Un estado fundamental H_O es una superposición de todos los estados posibles en la base propia de H_P [27] donde $x\ i$ es la puerta NOT sobre el i-ésimo qubit [2].

2.2 ESTADO DEL ARTE

Debido a la importancia y el reto que representa el problema de la mochila binaria, en los últimos años se han reportado un gran número de algoritmos que buscan su solución. Estos se agrupan en algoritmos exactos, algoritmos de programación dinámica, algoritmos basados en back-tracking (incluidos ramificación y poda), algoritmos metaheurísticos y recientemente algoritmos basados o que simulan lógica cuántica.

Entre los algoritmos metaheurísticos más destacados se encuentran los algoritmos genéticos, el recocido simulado, la optimización por enjambre de partículas (PSO), la búsqueda tabú [3], [10], el algoritmo evolutivo cuántico [11], el algoritmo genético cuántico [12] y el algoritmo VQE (Variational Quantum Eigensolver) [13].

2.2.1 Trabajos previos en el ámbito de algoritmos clásicos

Varios algoritmos propuestos en la literatura para resolver el problema de la mochila binaria tienen baja precisión y caen fácilmente en soluciones óptimas locales. Para superar estos problemas, en 2016 [31] se propone una versión binaria del algoritmo del mono (MA). Para validar la eficiencia del algoritmo propuesto, se realizan experimentos con varias instancias y se comparan los resultados con cinco algoritmos metaheurísticos reportados en la literatura (BPSO, MBPSO, NGHS, DGHS y S-bAFSA). Los experimentos muestran que el algoritmo CGMA propuesto tiene grandes ventajas en la resolución de problemas de la mochila binaria fijos y aleatorios y problemas de pequeña y gran escala. Dado que las pruebas de CGMA se realizaron contrastando los resultados de algoritmos de diferentes tipos y con una amplia variedad de instancias, CGMA se consideró en su fecha de publicación, una alternativa eficaz para resolver problemas binarios de la mochila.

[1] Matriz de 2x2 cuyo primo $\left[\left(\mathbb{I} + \sigma_i^z\right)/2\right]$ tiene vectores propios $|0, 1\rangle$ con valores propios 0,1 [27]

En 2017 [32] se propone una metaheurística hibrida para solucionar el problema de la mochila fuertemente correlacionada (SCKP). Se propone un algoritmo Hibrido de Optimización de Colonias de Hormigas (ACO) el cual combina el Sistema de Hormigas MAX-MIN y el Sistema de Colonias de Hormigas con el algoritmo 2-optimal, los cuales se ejecutan secuencialmente (la salida del primer algoritmo es la entrada del segundo). El algoritmo MMACS propuesto tiene como objetivo resolver óptimamente problemas SCKP, en caso de que no se encuentre una solución óptima, se utiliza el algoritmo 2-optimal; si la heurística 2-optimal no logra encontrar una solución óptima, al menos mejorará la calidad de la solución al reducir la brecha entre la solución encontrada y la óptima. Con este nuevo algoritmo se pretende mejorar las soluciones encontradas por el algoritmo ACO mediante el algoritmo 2-optimal para obtener mejores soluciones manteniendo un tiempo de ejecución reducido. El algoritmo propuesto se probó en un conjunto de instancias de alta y media dimensionalidad y se contrastó con el Algoritmo evolucionario inspirado en Cuántica (QEA). El algoritmo QEA no consigue encontrar soluciones apropiadas a los problemas. Por otro lado, el algoritmo propuesto es ineficiente en instancias de baja dimensionalidad, pero en la medida que la dimensionalidad crece encuentra soluciones óptimas. Este algoritmo propuesto tiene el problema de funcionar solo en instancias SCKP y no en otros tipos de problemas de la mochila.

Ese mismo año (2017) [33] se realiza el análisis de los algoritmos de Búsqueda tabú (Tabú Search, TS), Búsqueda Dispersa (Scatter Search, SS) y un algoritmo de búsqueda local (Local Search, LS). El objetivo del estudio fue determinar la eficiencia y precisión de cada uno en la solución del problema de la mochila binaria. Las pruebas se realizaron con el software HeuristicLab Framework, donde los algoritmos se comparan en función de la solución con mejor calidad y los tiempos de ejecución; medidos y comparados en un total de treinta observaciones tomando instancias de dimensionalidad media. Como resultado se tiene que el algoritmo SS registra la menor complejidad de tiempo (menor tiempo de ejecución) y el algoritmo TS consigue la menor desviación de la solución con la mejor calidad de la mochila. Como trabajos futuros, los autores plantean el uso de algoritmos metaheurísticos, como firefly, colonia de hormigas o GRASP, y comparar los resultados con más métricas y diferentes tamaños de muestra.

También en 2017 [34] se propone un algoritmo binario de araña social (BSSA) para resolver el problema de la mochila. El algoritmo propuesto combina con dos técnicas de manejo de restricciones para el problema de la mochila binaria. En BSSA se integra la exploración del algoritmo de araña social (SSA) y la explotación con un operador de reparación, además proponen dos técnicas de restricción basadas en el factor de penalización y una estrategia codiciosa para mejorar la eficiencia del algoritmo propuesto. Los resultados de la simulación en cinco instancias recientes según la literatura y con conjuntos de datos fuertemente correlacionados demuestran que el algoritmo propuesto tiene un rendimiento superior en comparación con un algoritmo genético y un algoritmo basado en PSO, sin

embargo, las pruebas no son lo suficientemente extensas para concluir la eficacia del algoritmo propuesto.

El algoritmo de optimización por enjambre de partículas binarias (BPSO) original y sus variantes no pueden proporcionar resultados totalmente satisfactorios debido al uso de funciones de desplazamiento inapropiadas, estas funciones no le proporcionan al algoritmo un buen equilibrio entre exploración y explotación en el espacio de búsqueda, lo que limita su desempeño. Para superar este problema, en 2017 [35] se propone agregar una función de desplazamiento variable en el tiempo denominada TVT-BPSO. Los resultados experimentales presentados demuestran que TVT-BPSO supera a las variantes de BPSO y a tres variantes de BPSO propuestas para solucionar el problema de mochila binaria utilizando instancias de baja dimensión, alta dimensión y un problema de Truss de 200 individuos. Los autores sugieren que TVT-BPSO puede escalar mejor a problemas combinatorios de alta dimensión que las variantes existentes y aunque no se evalúe la solución propuesta con diferentes implementaciones de otros algoritmos metaheurísticos, se sugiere que la solución propuesta puede lograr resultados competitivos o mejores.

En 2018 [36] se presenta un acelerador de hardware basado en FPGA para reducir el tiempo de procesamiento requerido para resolver problemas de la mochila binaria en instancias de baja, media y alta dimensionalidad utilizando el algoritmo de la Búsqueda Armónica Binaria (BHS). Los resultados experimentales revelaron una aceleración significativa en comparación con dos implementaciones de software paralelas del mismo algoritmo, así como una implementación de hardware que utiliza el Algoritmo de optimización por enjambre de partículas binarias (BPSO). Todos los resultados obtenidos por las distintas implementaciones son óptimos lo que indica que la calidad de las soluciones no se vio afectada, sin embargo, se observa una diferencia considerable en el tiempo de ejecución de BHS frente a BPSO.

También en 2018 [37] se presenta un nuevo algoritmo de optimización caótico basado en la mariposa monarca (CMBO). En el algoritmo de la mariposa monarca (MBO) se introduce la teoría del caos con el objetivo de acelerar la optimización y mejorar las capacidades de búsqueda global/local. Se utilizan doce mapas caóticos unidimensionales para ajustar los parámetros de CMBO y una mutación gaussiana con la cual se perturba una pequeña parte de las soluciones con peor aptitud. El desempeño de CMBO se verificó y analizó con tres grupos de instancias de problemas de la mochila binaria (no correlacionados, débilmente correlacionados y fuertemente correlacionados). Los resultados muestran que la introducción de un mapa caótico apropiado y la perturbación gaussiana pueden mejorar significativamente la calidad de la solución junto con el rendimiento general del algoritmo propuesto. El CMBO propuesto puede superar al MBO estándar y los algoritmos ABC, CS, DE, GA, FA, SFLA, HS y MBO, pero no se tiene en cuenta la dimensionalidad que se puede manejar en los diferentes problemas de mochila.

Algunos algoritmos metaheurísticos pueden fallar al quedar atrapados en un óptimo local, por ello en 2019 [38] se propone un algoritmo de optimización de ballenas basado en oposición (OWOA) que permita encontrar la solución a problemas de la mochila binaria. La oposición se realiza calculando el vector opuesto de una posible solución, por ejemplo, si un candidato es (001000101), el vector opuesto será (110111010). Se realizaron 3 experimentos para validar el algoritmo propuesto, en el primer experimento se comparó WOA con OWAO ejecutado la comparación 100 veces y encontrando el valor medio de los resultados, y en el segundo y tercer experimento se validó el rendimiento de OWOA comparando el resultado con los algoritmos HS-Jaya y CGMA utilizando 10 casos de problemas de mochila diferentes. Los resultados obtenidos muestran que en el valor medio de los resultados se nota una mejora notable en el rendimiento en comparación con OWOA. Como trabajos futuros se recomienda resolver diferentes problemas de optimización, como pruebas de software y problemas del vendedor viajero, y estudiar una nueva variante de OBL para mejorar WOA.

Los autores de otro estudio publicado en 2019 [39] se fijan el objetivo observar los efectos de los lobos dominantes en la eficiencia de la metaheurística de Optimización del Lobo Gris (GWO). Para probar el desempeño de los enfoques desarrollados y observar los efectos de los lobos dominantes modificados, se emplean tres conjuntos diferentes de evaluaciones comparativas que incluyen problemas continuos, combinatorios sin restricciones y con restricciones. Entre los problemas escogidos para probar el enfoque propuesto se utilizó el problema de la mochila binaria. Todas las modificaciones desarrolladas se comparan con el GWO estándar y el algoritmo de optimización por enjambre de partículas (PSO). El estudio experimental y los resultados verificados estadísticamente demuestran que los lobos dominantes en GWO tienen efectos cruciales en la eficiencia del GWO estándar. Además, las pruebas estadísticas demuestran que las modificaciones de GWO desarrolladas superan significativamente a algunos de los algoritmos informados en la literatura relacionada, como por ejemplo PSO. Para dar solución al problema de la mochila los artículos son ubicados en orden decreciente de acuerdo con la densidad de cada artículo (beneficio/peso), luego los artículos clasificados se asignan a la mochila hasta que se excede la capacidad de la mochila; este método de llenado podría dejar soluciones optimas por fuera del estudio.

En un trabajo reciente de 2021 [40] se propone un algoritmo de moho mucilaginoso binario mejorado (SMA) para resolver el problema de la mochila binaria. Este algoritmo utiliza una función de trasferencia para convertir las soluciones de un espacio de búsqueda continuo a un espacio de búsqueda binario, un operador de mutación gaussiana que permite aumentar la diversidad y evitar la convergencia excesiva durante el proceso de optimización, un operador de cruce que permite dispersar soluciones en el espacio del problema y salir de soluciones atrapadas en óptimos locales y una función de penalización y reparación que permiten convertir soluciones no factibles en factibles. El algoritmo propuesto se comparó contra 7 algoritmos (PSO, HHO, TSA, WOA, FFA, TLBO, AOA) y el proceso se realizó en

tres fases. Inicialmente se evaluaron instancias de 1 a 20 elementos, seguido de 21 a 45 elementos y finalmente con instancias de 46 a 63 elementos, con una métrica de evaluación enfocada en el tiempo de respuesta. En las dos primeras fases el algoritmo propuesto presenta una mejora significativa frente a sus contrapartes, y en la tercera fase a pesar de no encontrar la solución óptima en todos los escenarios, de manera general, presenta mejores tiempos de respuesta.

Un novedoso algoritmo de la libélula (Dragonfly Algorithm - DA) basado en la teoría de la alimentación de las libélulas y la evasión de los depredadores es propuesto en [41]. Este algoritmo tiene la capacidad de resolver problemas de optimización binaria debido a que introduce una un mecanismo de modulación de ángulo mejorado denominado IAMDA, el cual permite mejorar la estabilidad del algoritmo DA y la velocidad de convergencia. Se utiliza DA para evolucionar los coeficientes de una función trigonométrica la cual es el mecanismo de modulación que se utiliza para generar cadenas de bits, las cuales serán la solución al problema que se desea tratar. Para probar el rendimiento de IAMDA se contrasta con los algoritmos AMDA, BDA y BPSO, considerando 12 problemas de mochila binaria junto con 13 funciones de referencia clásicas. Los resultados experimentales demuestran que IAMDA tiene una velocidad de convergencia y una calidad de solución superiores. Como trabajos futuros se plantean probar el algoritmo propuesto con problemas de mochila binaria multidimensional o problemas multiobjetivo.

2.2.2 Trabajos previos en el ámbito de computación cuántica

En 2017 [2] se propone un algoritmo para solucionar el problema de la mochila con ganancias y pesos enteros, mediante computación cuántica adiabática. En este documento se realizan dos implementaciones, una utiliza un Ising Hamiltoniano que requiere n + c qubits para representar la solución y otra más eficiente la cual utiliza n + [log_2 C] + 1 qubits. En este documento se puede observar que reducir la cantidad de qubits aplicando una trasformación logarítmica tiene un impacto positivo en el rendimiento del algoritmo, ya que reduce considerablemente la cantidad de qubits necesarios para procesar una solución. En el trabajo presentado no se realizan pruebas comparativas detalladas con diferentes tipos de algoritmos del estado del arte, ya sean algoritmos clásicos, otras propuestas cuánticas puras o híbridos.

En 2019 [11] se propone un Algoritmo Evolutivo Genético Cuántico Mejorado (AEC-M) para solucionar el problema de la mochila binaria, el cual esta basado en la tecnología de catástrofe del ángulo de rotación dinámico donde se diseña un operador de puerta giratoria cuántica que ajusta de forma adaptativa los valores del ángulo de rotación de los qubits de acuerdo con el valor de aptitud y las generaciones de la evolución. Lo anterior se hace con el objetivo de que los spines de cada qubit del cromosoma apunten en la dirección de la solución. Se evalúa la solución propuesta con un algoritmo evolutivo cuántico (AEC) y un algoritmo evolutivo clásico AE. Los resultados experimentales permiten observar que (AEC-M) tiene un mejor rendimiento que AEC y que AE. Si se varia la cantidad de generaciones a evolucionar con respecto del valor de aptitud obtenido, se puede

observar que AEC con cantidades de generaciones a evolucionar pequeñas (1 - 100) arroja valores de aptitud estables, visualizándose una curva de convergencia muy suave hasta estabilizarse en un valle que varía muy poco; a diferencia de AEC-M el cual arroja mejores resultados de aptitud con un numero grande en la cantidad de generaciones a evolucionar, pero el crecimiento de la curva es muy brusco, con lo cual se observa que resultan óptimos locales muy a menudo.

En 2021 [42] se propone un algoritmo de evolución diferencial que combina algunos de los principios de la computación cuántica con un optimizador de lobo gris (QDGWO) para solucionar el problema de la mochila binaria. QDGWO combina los principios de superposición de la computación cuántica, las operaciones de evolución diferencial y los comportamientos de caza de los lobos grises. Con este algoritmo se espera mejorar el rendimiento en cuanto a diversidad y convergencia, y mejorar el rendimiento en instancias de alta dimensionalidad. Este algoritmo utiliza operaciones de mutación, cruce y observación cuántica para generar nuevos individuos de prueba. El optimizador de lobo gris adaptativo y la puerta de rotación cuántica se utilizan para preservar la diversidad de la población y acelerar la búsqueda de la solución óptima global en el caso de que los individuos de prueba sean peores que los individuos actuales. Con los resultados experimentales se logra observar las ventajas de la optimización colaborativa con operaciones de mutación adaptativa, cruce y puerta de rotación cuántica con el GWO adaptativo en la investigación del espacio de búsqueda.

Esta página ha sido dejada intencionalmente en blanco.

CAPÍTULO 3

3 MARCO DE TRABAJO PROPUESTO

3.1 METODOLOGÍA DE DESARROLLO

En este trabajo se utilizó el patrón de investigación iterativo (PII) propuesto por Pratt [46], para el desarrollo del marco de trabajo de comparación que se presenta en este capítulo, el algoritmo cuántico propuesto en el **Capítulo 4** y las tres (3) metaheurísticas del estado del arte que se presentan en el **Capítulo 5**. PII se compone de cuatro etapas principales, a saber: observación de campo (O), identificación del problema (I), desarrollo de la solución (D) y pruebas de campo (P). Los algoritmos se desarrollaron y refinaron a lo largo de 3 ciclos compuestos de estas etapas.

Durante la etapa de observación se realizó la identificación de los algoritmos que logran dar solución a problemas de optimización y que son apropiados para abordar el problema de la mochila binaria, cuáles son sus falencias y cuáles son las métricas de comparación de efectividad y eficiencia que permiten medir su desempeño. En todas las etapas de observación se identificó que es necesario contar con conjunto de datos extenso y diverso que permita contemplar varios escenarios de pruebas para determinar las fortalezas y debilidades de los algoritmos, esto brinda un acercamiento acerca de los problemas que puede afrontar cada algoritmo en un entorno real.

Luego, en la etapa de identificación se escogieron los algoritmos que se consideran relevantes para realizar la evaluación de soluciones cuánticas y se seleccionaron algunas métricas que resultan ser constantes en la evaluación de los algoritmos propuestos que se estudiaron. Por otra parte, para abordar la complejidad que resulta de la selección o construcción de los conjuntos de datos, se decidió tomar como insumo tres generadores (comentados en este capítulo) automáticos de instancias de mochila los cuales permiten construir los datos necesarios de acuerdo con algunos parámetros de diversidad definidos previamente.

Las etapas de desarrollo y pruebas (experimentación) se realizaron en paralelo, en un proceso de retroalimentación constante. Este proceso se ejecutó de la siguiente manera: Para el marco de trabajo se implementó una versión inicial del módulo principal y el módulo generador, con ello se buscó realizar la integración de uno de los tres programas ejecutables seleccionados para la generación del conjunto de datos de prueba; una vez se obtuvieron los datos de prueba iniciales se procedió a implementar el módulo de archivos y finalmente se inició la construcción del módulo de algoritmos teniendo presente que se buscaba contar con un marco de trabajo extensible y sencillo con un buen nivel de modularidad que permita realizar

modificaciones sin afectar el funcionamiento del programa. En la medida que se avanza en la ejecución de cada ciclo se van integrando los dos programas faltantes que realizan la generación del conjunto de datos, se adiciona un componente para realizar la gestión de los argumentos introducidos por la línea de comandos y se implementan los algoritmos metaheurísticos que se usarán para comparar la propuesta cuántica. En paralelo con los ciclos de PII se realizaron experimentos para evaluar y contrastar la calidad de los resultados y los tiempos de ejecución de los algoritmos implementados en el marco de prueba, junto con el algoritmo cuántico.

En forma transversal al desarrollo de las anteriores actividades se realizaron actividades de documentación, escritura de la monografía y sus anexos; Para finalizar, en el último ciclo de este proceso se realizó la escritura de un artículo para la divulgación de los resultados obtenidos.

Durante la última etapa se terminó de ajustar completamente los pseudocódigos de los algoritmos y se realizó un proceso retrospectivo para identificar los aspectos necesarios para lograr una adecuada repetibilidad de los resultados de las pruebas finales que se presentan en el **Capítulo 5**.

3.2 DESCRIPCIÓN DEL MARCO DE TRABAJO

En la actualidad, la mayoría de las soluciones cuánticas a problemas de optimización se realizan utilizando un modelo de circuitos cuánticos que trabajan con programación a bajo nivel, es decir con compuertas cuánticas que alteran el estado de un qubit sobre una máquina que trabaja bajo las leyes de la mecánica cuántica. Esta tecnología permite realizar cálculos en paralelo a través del principio de superposición de estados.

Debido a que esta tecnología aún está en una fase temprana de desarrollo, aun no se cuenta con lenguajes de programación de alto nivel y se presentan algunos problemas de escalabilidad ya que con el incremento del espacio de búsqueda de un algoritmo aumenta la cantidad de qubits requeridos para realizar operaciones de cálculos complejos. Debido a esto y mientras se realizan avances significativos en materia de Hardware y Software, se plantea el uso de emuladores de computación cuántica que incorporan la lógica de la mecánica cuántica para desarrollar heurísticas que permitan dar soluciones eficientes a problemas complejos sobre una infraestructura de computación clásica.

Por lo anterior, en el presente trabajo se utilizan algoritmos de computación cuántica basado en un emulador que ejecuta programas en Python soportados en una librería denominada Qiskit, que ha sido desarrollada por el equipo de IBM para el estudio de soluciones cuánticas. Esta librería contiene un conjunto de herramientas y algoritmos desarrollados con base en la experimentación que ha tenido IBM en computadores cuánticas reales, y cuenta con una base teórica robusta, una documentación extensa y detallada, y además con el apoyo de la infraestructura que IBM ha venido creando en los últimos años en este interesante tema.

En el proceso de revisión del estado del arte se logró observar un amplio abanico de algoritmos que se han venido utilizando para dar solución al problema de la mochila binaria, donde una de las preocupaciones principales consiste en tratar de encontrar un equilibrio entre la exploración y la explotación del espacio de búsqueda (que crece exponencialmente con el número de elementos o ítems) y lograr reducir el tiempo de respuesta utilizado como métrica de comparación de eficiencia y eficacia, además de la calidad de las soluciones encontradas. Entre los algoritmos del estado del arte propuestos y la escogencia de los algoritmos que son de interés para el presente trabajo, se logra apreciar un uso recurrente de algoritmos metaheurísticos poblacionales y se plantea la necesidad de realizar estudios comparativos con instancias de prueba que tengan diversidad en cuanto a la correlación de los pesos y valores (profit) de los elementos (objetos) y la dimensionalidad del problema (número de elementos). Esto ayuda a definir algunos criterios para la selección de los algoritmos que serían objeto de comparación en el presente trabajo.

En la **Figura 1** se muestra un diagrama de clases de alto nivel del marco de trabajo (framework) de prueba implementado. Las clases de color naranja corresponden al núcleo del marco de trabajo, y las clases de color azul son clases asociadas a la implementación de un algoritmo específico o implementaciones de clases abstractas y los artefactos en color verde corresponden a los archivos ejecutables que contienen los algoritmos generadores de los conjuntos de datos. Por último, los artefactos de color rojo hacen referencia a documentos de respuesta con los resultados obtenidos de la evaluación de los algoritmos y a los archivos resultantes de la generación del conjunto de datos.

El marco de trabajo de divide en cuatro módulos:

- **Modulo principal:** Se encarga de controlar el funcionamiento del framework, definiendo la lógica requerida para crear y realizar pruebas sobre múltiples algoritmos y conjuntos de datos.
- **Modulo generador:** Se encarga de generar los conjuntos de datos de acuerdo con los parámetros ingresados por la línea de comandos.
- **Módulo de archivos:** Se responsabiliza de cargar los conjuntos de datos con los cuales se realiza la comparación de rendimiento de los algoritmos y de escribir los resultados en un documento de texto.
- **Módulo de algoritmos:** Define la estructura base para lograr integrar nuevos algoritmos diseñados para resolver el problema de la mochila binaria.

A continuación, se detalla cada uno de los módulos mencionados.

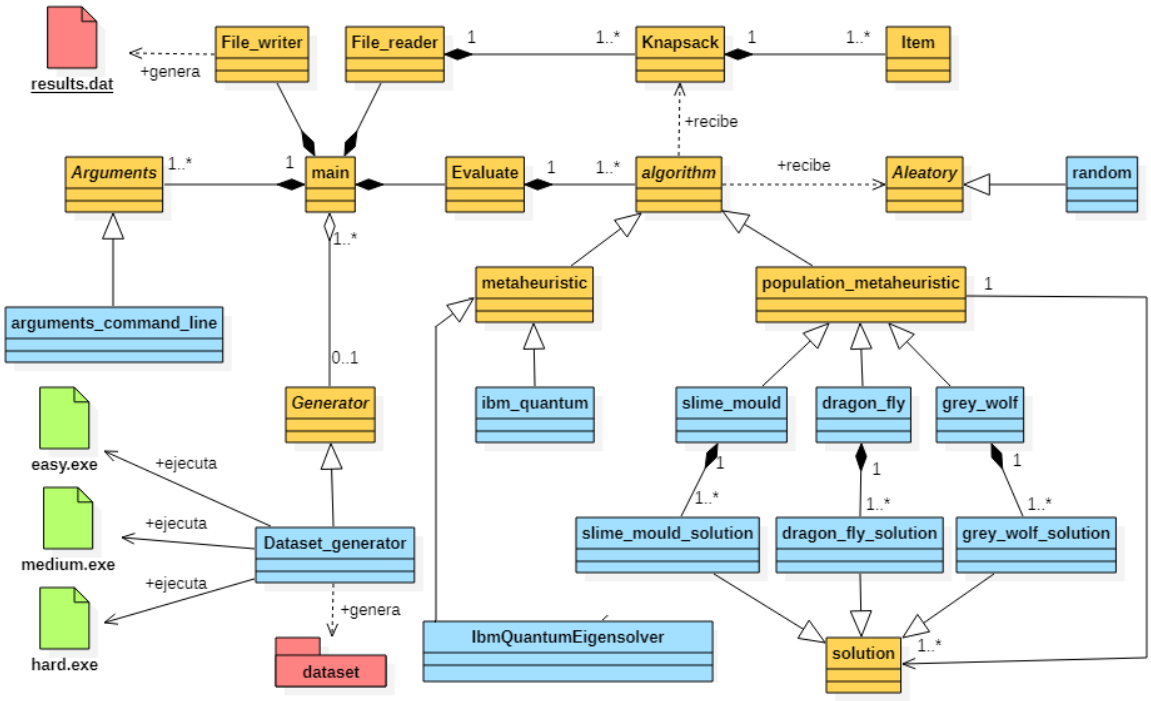

Figura 1. Diagrama de clases de alto nivel

3.2.1 Módulo principal

En la **Figura 2** se muestran las clases que hacen parte del módulo principal. Este módulo se puede subdividir en dos partes, por un lado, se cuenta con las clases encargadas de la gestión de los argumentos ingresados por la línea de comandos, y por otro lado la clase principal (main) y de evaluación encargada de gestionar el funcionamiento del marco de trabajo de prueba.

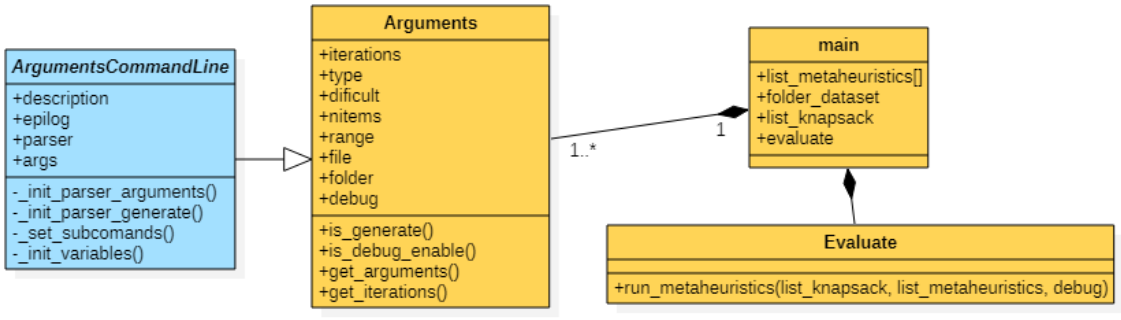

Figura 2. Diagrama de clases detallado del módulo principal

3.2.1.1 Submódulo de argumentos

En este submódulo se implementa una clase abstracta para que el aplicativo tenga la capacidad de ser extensible y con ello se puedan agregar las clases concretas

que se consideren pertinentes dependiendo del modelo de argumentos que se desee utilizar. La clase abstracta **Arguments** contiene cuatro métodos básicos para la gestión de los argumentos ingresados en la ejecución del programa, estos son:

- **is_generate():** Indica si se debe generar un nuevo conjunto de datos.
- **is_debug_enable():** Indica si el usuario desea que el programa se ejecute en modo de depuración, esto habilita la salida de texto que permite la visualización de las operaciones que se están ejecutando.
- **get_arguments():** Retorna una instancia de tipo **ArgumentParser** la cual contiene los argumentos ingresados por el usuario en la línea de comandos al momento de la ejecución del programa.
- **get_iterations():** Retorna la cantidad de iteraciones que definió el usuario. Con este valor se define cuantas veces se debe ejecutar cada algoritmo, también se conoce como el número máximo de repeticiones de un experimento, donde un experimento es igual a un argumento resolviendo un problema concreto con unos parámetros específicos y una semilla de inicialización de números aleatorios especifica que permita asegurar la repetibilidad del experimento.

Debido a que el programa se ejecuta por la línea de comandos, se tiene la clase concreta **Arguments_command_line** la cual se encarga de implementar los métodos heredados de la clase padre y además de eso realiza la configuración del menú de ayuda para el usuario, en este menú se tiene información básica de los argumentos que son admitidos en el proceso de ejecución del aplicativo.

Los argumentos opcionales de esta clase son:

- **[-h, --help]:** Permite visualizar el mensaje de ayuda.
- **[-i ITERATIONS, --iterations ITERATIONS]:** Con esta bandera se permite indicar el número de iteraciones que se debe utilizar al momento de ejecutar cada algoritmo con cada archivo de mochila disponible. Por defecto se toma el valor 31.
- **[-d, --debug]:** Le indica al programa que habilite la salida de texto para permitir observar una trazabilidad clara del cada fase del algoritmo.
- **[-fl FILE, --file FILE]:** Con esta bandera se permite indicar el nombre o ubicación de un archivo con la información de una mochila a evaluar.
- **[-fd FOLDER, --folder FOLDER]:** Con esta bandera se permite indicar el nombre o ubicación de una carpeta que almacene archivos con la información de las mochilas que se desean evaluar.

Subcomandos:

- **{generate}:** Subcomando encargado de realizar la gestión de las opciones para la generación de un conjunto de datos. Se explica en detalle en el apartado correspondiente al módulo generador.

3.2.1.2 Submódulo principal (main)

Las clases **Main** y **Evaluate** se encargan de orquestar todo el funcionamiento del marco de trabajo, entre sus funciones principales se destacan:

- Gestionar la generación u obtención del conjunto de datos necesario para realizar la evaluación.
- Coordinar la ejecución de todas las instancias de los algoritmos seleccionados de acuerdo con los parámetros ingresados por el usuario.
- Gestionar los algoritmos que serán objeto de pruebas.

3.2.2 Módulo generador

En la **Figura 3** se muestra el módulo generador encargado de generar el conjunto de datos pertinentes de acuerdo con la configuración ingresada por el usuario, en ella se puede observar que se relacionan tres artefactos, **easy.exe**, **médium.exe** y **hard.exe**, estos artefactos son programas generadores de instancias de mochila con los cuales se permite crear los conjuntos de datos necesarios. Los problemas (conjuntos de datos) se almacenan en la carpeta definida por el usuario.

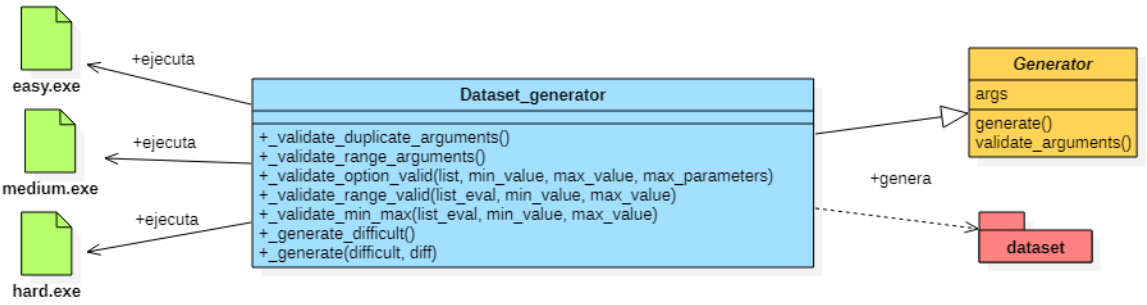

Figura 3. Diagrama de clases detallado del módulo generador

3.2.2.1 Conjunto de datos

Para la realización de las pruebas es necesario contar con un conjunto de datos extenso y diverso, por ello se decidió tomar tres generadores de instancias del problema de la mochila binaria ampliamente reconocidos por la comunidad científica que se encuentran disponibles en [43]. Estos generadores están desarrollados en el lenguaje de programación C y con ellos se tiene la posibilidad de generar instancias con distintas configuraciones para realizar pruebas de algoritmos en condiciones más realistas. Los generadores se detallan a continuación:

- **Easy.exe:** Genera instancias de prueba fáciles de resolver, este generador se describe en [44].
- **Medium.exe:** Generador avanzado para construir instancias de prueba con mediana dificultad, se consideran 14 tipos de instancias diferentes. este generador se describe en [45].
- **Hard.exe:** Generador para construir instancias de prueba complejas, este generador se describe en [9].

Para la construcción del conjunto de datos se definieron cuatro variables para tener en cuenta:

- **Tipo de correlación:** Hace referencia al tipo de correlación que deben tener los elementos de la mochila (no correlacionado, débilmente correlacionado, fuertemente correlacionado y con pesos y valores iguales).
- **Dificultad:** Hace referencia a la dificultad que debe tener el conjunto de datos, si se requiere una dificultad normal, el conjunto de datos se genera con el artefacto ***easy.exe***, si se requiere una dificultad media el conjunto de datos se genera con el artefacto ***medium.exe*** y si se requiere una dificultad alta el conjunto de datos se genera con el artefacto ***hard.exe****.*
- **Cantidad de elementos:** Hace referencia a la cantidad de elementos (objetos o ítems) que se van a crear para el problema.
- **Rango de los coeficientes:** Hace referencia al valor máximo permitido para los valores de peso de cada elemento disponible para almacenar en la mochila.

Teniendo en cuenta las variables definidas anteriormente, los archivos se almacenan en una carpeta general nombrada de la siguiente manera: *generated_dataset_[datetime]*, donde *datetime* hace referencia a la estampa de tiempo del momento en el que se está ejecutando el algoritmo; Dentro de esa carpeta se crean algunas subcarpetas que serán nombradas de acuerdo con el tipo de dificultad seleccionada y almacenaran los archivos que contienen toda la información de cada mochila que se generó. Adicionalmente cada archivo tendrá una denominación única que contiene los parámetros de creación y se rige bajo la siguiente nomenclatura: **`t[]_d[]_n[]_r[].dat`**. La información resultante que se genera en cada documento con los artefactos ejecutables contiene la estructura presentada en la **Figura 4**. Donde **n** indica la cantidad de elementos, **C** indica la capacidad máxima de la mochila**, p[0]** indica el beneficio del primer elemento, **w[0]** indica el peso del primer elemento y así sucesivamente hasta el elemento n-1.

3.2.2.2 Generación del conjunto de datos

Para habilitar esta funcionalidad es necesario indicarle al programa mediante la línea de comandos que debe realizar la generación. Este procedimiento se realiza enviando el subcomando {generate} en la ejecución del programa. Cuenta con los siguientes argumentos opcionales:

- **[h, --help]]:** Permite visualizar el mensaje de ayuda.
- **[-t TYPE [TYPE ...], --type TYPE [TYPE ...]]:** Indica el tipo de correlación entre los elementos de la mochila. Opciones: ([1=uncorrelated, 2=weakly correlated, 3=strongly correlated, 4=subset sum]).
- **[-d DIFFICULT [DIFFICULT ...], --difficult DIFFICULT [DIFFICULT ...]]:** Indica la dificultad del conjunto de datos a generar. Opciones: ([1=Easy, 2=Medium, 3=Hard]).
- **[-n NITEMS [NITEMS ...], --nitems NITEMS [NITEMS ...]]:** Indica la cantidad de elementos almacenados en cada mochila. El valor máximo es 13. Si se envían dos argumentos se tomará como un rango, si se envían más de dos se tomará como una lista. [{n1-n2} <> {n1, n2, ..., n}].
- **[-r RANGE [RANGE ...], --range RANGE [RANGE ...]]:** Indica el rango de creación de los pesos del conjunto de datos almacenado en la mochila. El valor máximo es 100. Si se envían dos argumentos se tomará como un rango, si se envían más de dos se tomará como una lista. [{n1-n2} <> {n1, n2, ..., n}].

```
n C
p[0] w[0]
p[1] w[1]
:
p[n-1] w[n-1]
```

Figura 4. Estructura del archivo con una instancia de la mochila

3.2.3 Módulo de archivos

En la **Figura 5** se muestra el módulo de archivos encargado de gestionar el acceso a los archivos necesarios en la ejecución del algoritmo. Este módulo consta de dos clases principales, una encargada de la lectura de los archivos y otra encargada de la escritura de los resultados en un archivo de salida.

3.2.3.1 FileReader

Esta clase se encarga de realizar la lectura de un archivo que contiene la información de un problema de la mochila binaria. Al momento de realizar la lectura se va construyendo una instancia de la clase **Knapsack** la cual almacena una lista que contiene instancias de la clase **Item** y en conjunto generan una abstracción de la mochila.

Debido a que se trabaja con instancias de baja dimensionalidad, si el archivo no contiene la descripción de la solución óptima global, esta se calcula por medio de un algoritmo de fuerza bruta que evalúa todas las posibles combinaciones y entre ellas selecciona la mejor. A continuación, se escribe el valor objetivo (fitness óptimo) en el archivo seguido de la solución óptima (elementos que van con un uno o con

un cero en la mochila). Al finalizar la lectura del archivo, en él se organizan los datos conforme a la **Figura 6**, donde **n**, **C**, **p[*]** y **w[*]** significan lo ya explicado previamente, **obj** indica el valor objetivo de la solución y al final del documento se encuentra la **solución óptima** al problema en cuestión la cual consta de una lista de unos o ceros donde uno significa que el elemento va en la mochila y cero el caso contrario.

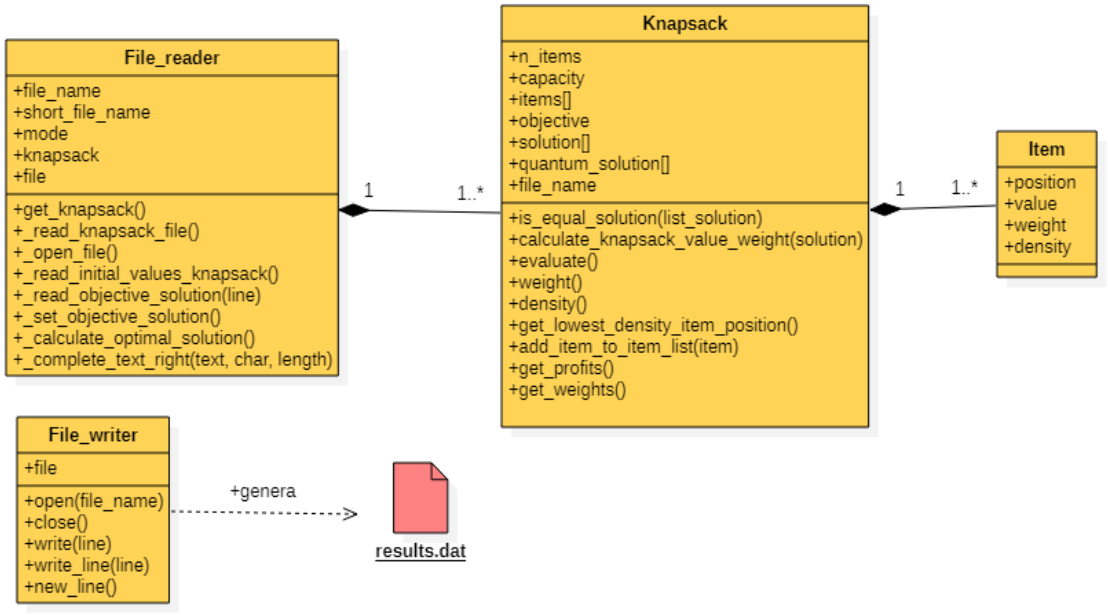

Figura 5. Diagrama de clases detallado del módulo de archivos

```
n C
p[0] w[0]
p[1] w[1]
:
p[n-1] w[n-1]
obj
0 0 0 … n-1
```

Figura 6. Estructura extendida del archivo con una instancia de la mochila

Terminado el proceso de lectura, la clase **FileReader** ha construido una instancia de la clase **Knapsack** con toda la información que contiene el archivo; esta instancia de la mochila se puede obtener invocando el método **get_knapsack()**.

3.2.3.2 FileWriter

Esta clase se encarga de realizar la escritura de los resultados de la ejecución de todas las iteraciones (repeticiones) de los algoritmos seleccionados en dos archivos de texto para su posterior evaluación. Estos archivos se almacenan en el fichero

solutions que se encuentra ubicado en el directorio raíz del proyecto y se nombran automáticamente de la siguiente manera:

- **result_[datetime].txt:** El primer valor que se registra en este documento es el número de iteraciones, seguido de los nombres de cada algoritmo ejecutado y a modo de encabezado, se registran las variables que fueron seleccionadas para realizar la evaluación de resultados. En la primera ejecución se registran los datos obtenidos de la ejecución de los algoritmos con cada problema de mochila, los campos que se registran son los siguientes: Nombre del archivo, Numero de ítems, Capacidad de la mochila, Valor objetivo (fitness optimo) y Solución óptima.

  Luego de registrar la información de la mochila evaluada, en la misma línea del archivo se registran los siguientes datos de respuesta obtenidos en el proceso de ejecución de cada algoritmo: Tasa de éxito, Fitness promedio, Desviación estándar, Valor del mejor (mayor) fitness, Valor del peor (menor) fitness y Tiempo promedio.

- **result_[datetime]_fitness.csv**: Al inicio de este archivo se registra el número de iteraciones y a modo de encabezado se registran los nombres de los algoritmos seleccionados para realizar el proceso de evaluación. En la primera ejecución, en una sola línea se registra el nombre de la mochila evaluada seguido por el promedio de los fitness obtenidos en la ejecución de todas las iteraciones de cada algoritmo.

La marca **[datetime]** equivale a una estampa de tiempo del momento en el que se creó el documento. Se agrega para evitar errores por duplicidad de nombres y brindar un identificador único para cada archivo de resultados.

3.2.4 Módulo de algoritmos

En la **Figura 7** se muestra el módulo de algoritmos, este módulo se encarga de brindar una arquitectura base de alto nivel para la implementación de nuevos algoritmos, la cual busca que se puedan adicionar nuevas metaheurísticas sin que se afecte el funcionamiento del marco de trabajo.

3.2.4.1 Algorithm

Esta clase abstracta se encarga de definir el método de ejecución que deben implementar las clases concretas, además define las variables globales que servirán como parámetros de ejecución de cada algoritmo, entre ellas se define una variable para el problema de la mochila que se va a resolver, una variable que define el mecanismo de aleatoriedad que se desea utilizar y una variable donde se almacena la mejor solución encontrada.

Además de considerar una clase para definir una metaheurística general denominada **Metaheuristic**, la escogencia de los algoritmos que van a ser objeto de estudio requiere que se tenga una metaheurística especifica basada en

población denominada **PopulationMetaheuristic**, con ella se permite agregar algoritmos que simulan metaheurísticas poblacionales o de enjambre.

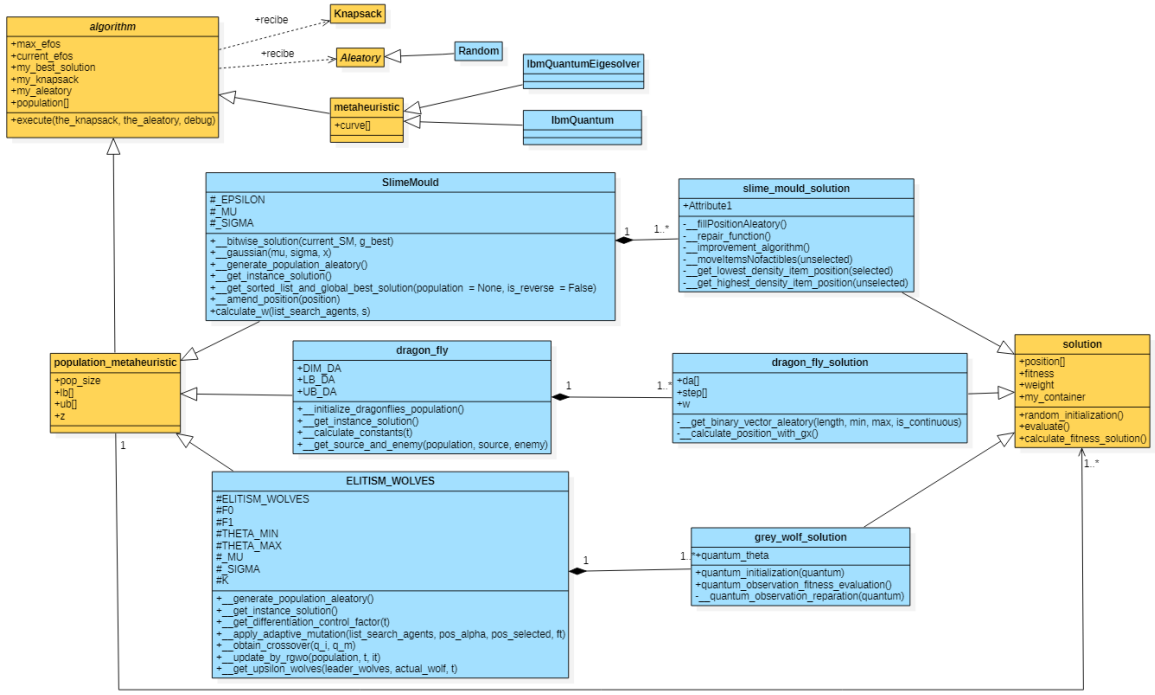

Figura 7. Diagrama de clases del módulo de algoritmos

En la definición del método abstracto *execute(the_knapsack, the_aleatory, debug)* se recibe una instancia de la clase **Knapsack** y **Aleatory** respectivamente, por medio de estos parámetros se realiza la ejecución del algoritmo especifico utilizando la instancia de mochila deseada y una estrategia de aleatoriedad. La implementación de nuevos algoritmos con base en este contrato le permite al marco de trabajo tener la capacidad de adaptarse a diferentes lógicas de ejecución manteniendo un alto nivel de extensibilidad.

3.2.4.2 PopulationMetaheuristic

Para realizar la comparación del marco de trabajo con los algoritmos seleccionados, se hace necesario la creación de esta clase abstracta que hereda de la clase **Algorithm** y permite agregar algoritmos que se desarrollan bajo los lineamientos de metaheurísticas poblacionales o de enjambre. En esta clase se adiciona un atributo que permite controlar la cantidad de individuos que se van a generar en cada iteración.

3.2.4.3 Metaheuristic

Esta clase Abstracta que hereda de la clase **Algorithm,** brinda una capa de abstracción adicional que define el método principal de ejecución que deben implementar las clases concretas, con esto se garantiza que el marco de trabajo

quede abierto a la inclusión de cualquier tipo de algoritmo adicional sin que se deban realizar cambios drásticos que afecten su funcionamiento.

3.2.4.4 Solution

Esta clase abstracta define los métodos necesarios para realizar las operaciones que requiera cada algoritmo concreto, tiene una relación de dependencia con la clase **Algorithm** ya que mediante esta clase se logra realizar operaciones de cruce, mutación, evaluación, cálculo de fitness entre otras, a las instancias de solución evaluadas en cada iteración. Esta clase define dos constructores para su inicialización, un constructor se encarga de inicializar una instancia vacía y otro constructor permite realizar una copia de una solución previamente instanciada.

3.2.4.5 IbmQuantum

En esta clase de encuentra el desarrollo de toda la lógica de simulación cuántica que se encarga de calcular los operadores de Pauli mediante la definición de la **Ecuación 11**, estos operadores sirven de insumo para la clase **ExactEigensolver** la cual es la instancia de Qiskit Aqua encargada de encontrar la solución al problema de mochila.

3.2.4.6 IbmQuantumEigensolver

En esta clase se sobrescribe el método *execute(the_knapsack, the_aleatory, debug)* heredado de la clase padre **Metaheuristic**, en ella se define el algoritmo que da solución al problema específico de la mochila utilizando las herramientas de la librería Qiskit en su versión 0.34.2. La solución se determina hallando los vectores propios del problema en cuestión utilizando la clase **MinimumEigenOptimizer()** y enviando como parámetro una instancias de la clase *NumPyMinimumEigensolver* que desempeña el papel de solucionador de vectores propios.

Las clases restantes marcadas en color azul corresponden a las implementaciones de los algoritmos del estado del arte seleccionados para realizar el estudio comparativo que se explican más adelante.

CAPÍTULO 4

4 PROPUESTA CUÁNTICA

4.1 MAPEO DE UNA FUNCIÓN DE COSTO A UN HAMILTONIANO CUÁNTICO

Para dar solución al problema de la mochila binaria mediante el modelo de computación cuántica, en [27] se realiza la formulación de Ising de los 21 problemas NP-completos de Karp. Entre las formulaciones presentadas se tiene una primera aproximación de la formulación de un Hamiltoniano Ising para el problema de la mochila binaria con una ecuación que consta de n + C qubits la cual permite representar y orientar la búsqueda de una solución. Dado que la cantidad de qubits en los sistemas cuánticos actuales tiende a ser un factor de importancia, se hace necesario que en los algoritmos se logre reducir al máximo los qubits requeridos, por ello en [2] se realiza la reducción del vector solución de una cantidad N de qubits a log_2 N. Los dos modelos de solución se detallan a continuación ya que podrían servir como base para lograr resolver problemas de optimización similares.

4.1.1 Solución con (n + C) qubits

Teniendo en cuenta la definición del problema de la mochila propuesta en la **Ecuación (6)** y trabajando con la versión binaria de la misma, se tienen *n* variables x_i donde *1 ≤ i ≤ n*, las cuales denotan valores binarios que se fijan con uno (1) si un determinado elemento debe ir en la mochila o cero (0) en caso contrario, y *j* variables y_j donde *1 ≤ j ≤ C* que denotan una variable binaria la cual es 1 si el peso de la mochila es j, o cero (0) en caso contrario; Así, la cantidad de qubits requerida para modelar la solución resulta de la suma del número de artículos (elementos u objetos) y la capacidad máxima de la mochila: (*n* + C) → $[x_1, \dots, x_n, y_1, \dots\ y_C]$.

La solución al problema consiste en construir un modelo de Ising H = H_A + H_B, con H_A como el término que controla el cumplimiento o no de la restricción del problema, cómo se define en la **Ecuación (6).**

$$H_A = A\left(1 - \sum_{j=1}^{C} \quad y_j\right)^2 + A\left(\sum_{k=1}^{C} \quad ky_k - \sum_{i=1}^{n} \quad W_i x_i\right)^2 \qquad (6)$$

Con H_A se obliga a que el peso solo pueda tomar un valor y que el peso de los objetos en la mochila sea igual al valor que se desea. Por ejemplo, si una solución da un valor igual a 10 para la variable peso, solo $y_{10} = 1$, el resto de $y_k = 0$ para los otros valores de k. Si la capacidad de la mochila (C) fuese de 20, esta solución es factible y en ese caso H_A sería igual a $A(1-1)^2 + A(10-10)^2 = 0$, eliminando así a H_A. Pero si C tuviera u valor igual a 8, esta solución no sería factible y en este

caso H_A sería igual a $A(1-0)^2 + A(0-10)^2 = 101A$ haciendo que el valor de H_A sea grande (positivo, alejando la solución del mínimo), y entre más se supere el peso máximo de la mochila, mayor será el valor de H_A. Finalmente se tiene H_B en la **Ecuación (7)**.

$$H_B = -B\sum_{i=1}^{n} P_i x_i \qquad (7)$$

El término H_B sirve para maximizar el beneficio (a mayor beneficio más se acerca al optimo o mínimo del Hamiltoniano) y el término H_A asegura que se cumpla la restricción de peso total. Debido a que se requiere que no sea posible encontrar una solución donde H_A no se cumpla a expensas de que H_B se vuelva más negativo, se requiere fijar la condición: 0 < B * Max (P_i) < A [27].

4.1.2 Solución con (n + [log₂ C] + 1) qubits

Realizando una modificación a la solución planteada anteriormente, la cual requiere (*n* + C) qubits, se puede reducir drásticamente la cantidad de giros y_j adicionales que deben agregarse. Así, codificando la magnitud del peso máximo (C) como un valor binario, se define una variable entera M tal que $2^M \leq C < 2^{M+1}$, con M = $\lfloor C \rfloor$; De esta manera solo se necesitan (M+1) variables binarias: [y_0 ... y_M], en lugar de C variables binarias: [y_1 ... y_C], para codificar una variable que puede tomar C valores (C >> M).

Aplicando la nueva configuración, para resolver el problema de la mochila, se requieren (n+$\lfloor C \rfloor$+1) qubits [27]. En la **Ecuación (8)** se presenta H_A construido de acuerdo a la nueva representación de y_j [2].

$$H_A = A\left(\sum_{j=0}^{M-1} 2^j y_j + (C + 1 - 2^M)y_M - \sum_{i=1}^{n} W_i x_i\right)^2 \qquad (8)$$

Si se tiene una solución con un peso de 10 y una capacidad máxima *C* = 20, entonces la cantidad de qubits requeridos para representar el peso de la solución es M = $\lfloor 20 \rfloor$+1=5, por lo tanto, se tiene el vector $y = [0\ 1\ 0\ 1\ 0]$ = 10, que corresponde a la representación de 10 en binario de izquierda a derecha; Así, se cumple la condición $16 \leq C = 20 < 32$. Esta solución es factible, por lo que $H_A = A\ (10 + (20 + 1 - 32)0 - 10)^2 = 0$, quedando eliminado H_A. Por otro lado, si *C* = 9, se tendría el vector $y = [0\ 1\ 0\ 1] = 10$ con $H_A = A\ (2 + (9 + 1 - 8)1 - 10)^2 = 36\ A$, que representa un valor alto de contrapeso ya que se supera la capacidad máxima permitida. Este valor crecerá entre más se rebase la capacidad de la mochila.

Este segundo hamiltoniano es el que se toma como referente en el presente trabajo como solución del problema de la mochila binaria. En resumen, el Hamiltoniano H = H_A + H_B esta dado por la **Ecuación (9)**, que utiliza una representación en base cero (0) para los vectores.

$$H = A\left(\sum_{j=0}^{M-1} 2^j y_j + (C + 1 - 2^M)y_M - \sum_{i=0}^{n-1} W_i x_i\right)^2 - B\sum_{i=0}^{n-1} P_i x_i \qquad (9)$$

Los diferentes valores implicados se representan de la siguiente manera [13]:

- La solución se representa como un vector $\vec{X}$ = [x_0 ... x_{n-1}] ∈ {0, 1} en donde cada elemento se representa en un índice del vector. Si el valor es 1, entonces el elemento se almacena en la mochila y si es cero (o) el elemento queda fuera.
- El peso de la solución se representa en binario de izquierda a derecha como un vector $\vec{Y}$ = [y_0 ... y_m] ∈ {0, 1}.
- El valor del peso de los elementos se representa como un vector W= [W_0 ... W_{n-1}].
- El beneficio de los elementos se representa como un vector P = [P_0 .. P_{n-1}].
- La variable C representa la capacidad de la mochila.
- La variable A representa un número lo suficientemente grande para dominar la multiplicación del valor B por la suma de los pesos de todos los elementos.

Para construir una función de costo C($\vec{K}$) se debe tener en cuenta que se desea maximizar la **Ecuación (9)** mientras se cumple con su respectiva restricción [13]. De la definición de H, se tienen M+1 nuevas variables binarias que se representarán en el vector y = [y_0 … y_M], donde los elementos pueden tomar valores diferentes de cero simultáneamente [2]. La solución en la función de costo se representará en un vector K compuesto por los vectores $\vec{X}$ y $\vec{Y}$, con $\vec{K}$ = [x_0 … x_{n-1}, y_n … y_{n+M-1}]. En la **Ecuación (10)** se presenta la función de costo C(K) definida por [13] y su polinomio expandido.

Se puede observar que para llevar el primer término de la **Ecuación (10)** a cero, se debe cumplir que $C \geq \sum_{i=0}^{n-1} W_i x_i$ y se tenga un S = $C - \sum_{i=0}^{n-1} W_i x_i$. El valor de L debe dominar la suma de los valores de P, esto es, $L > \sum_{i=0}^{n-1} P_i$. Si por el contrario $C < \sum_i W_i x_i$, el primer término nunca será cero que multiplicado por L evitará seleccionar soluciones no factibles. El valor mínimo de la función de costo será uno donde la restricción es respetada y la suma de los valores se maximice [13].

$$C(\vec{K}) = L\left(C - \sum_{i=0}^{n-1} W_i x_i - S\right)^2 - \sum_{i=0}^{n-1} P_i x_i;\ donde\ S = \sum_{j=n}^{n+M-1} 2^j y_j$$
$$C(\vec{K}) = L\left(C^2 + \left(\sum_{i=0}^{n-1} W_i x_i\right)^2 + S^2 - 2C\sum_{i=0}^{n-1} W_i x_i - 2CS + 2S\sum_{i=0}^{n-1} W_i x_i\right) - \sum_{i=0}^{n-1} P_i x_i \qquad (10)$$

4.1.3 Mapeo de una función de costo a su hamiltoniano cuántico

Al realizar el mapeo de las variables x_i por ($\mathbb{I}$ - Z_i)/2, y_j por ($\mathbb{I}$ - Z_j)/2 y eliminar el primer término que es constante, se obtiene la **Ecuación (11)**, donde $\mathbb{I}$ es una matriz identidad de orden *n+m* y Z_i es la matriz resultante de tensorizar *n+m* matrices donde en cada posición diferente de *i* los factores del tensor son matrices identidad y la posición *i* es una matriz de Pauli Z.

La matriz de Pauli se compone de dos vectores, un vector X y un vector Z, para este caso se modifica el vector Z únicamente. Para hallar los coeficientes de la matriz se debe resolver cada sumatoria resultante en la **Ecuación (11)**, encontrar el respectivo índice para cada valor de la sumatoria e ir adicionándolo a una lista. Por ejemplo, para el termino: $\sum_{i=0}^{n-1} \quad W_i\left(\frac{I-Z_i}{2}\right) \times \sum_{i=0}^{n-1} \quad W_i\left(\frac{I-Z_i}{2}\right)$, se muestra su algoritmo en la **Figura 8**.

$$
\begin{aligned}
C(\vec{K}) = L\Bigg(&\left(\sum_{i=0}^{n-1} \quad W_i\left(\frac{I-Z_i}{2}\right) \times \sum_{i=0}^{n-1} \quad W_i\left(\frac{I-Z_i}{2}\right)\right) \\
&+\left(\sum_{j=n}^{n+M-1} \quad 2^j\left(\frac{I-Z_j}{2}\right) \times \sum_{j=n}^{n+M-1} \quad 2^j\left(\frac{I-Z_j}{2}\right)\right) - 2C\sum_{i=0}^{n-1} \quad W_i\left(\frac{I-Z_i}{2}\right) \\
&-2C\sum_{j=n}^{n+M-1} \quad 2^j\left(\frac{I-Z_j}{2}\right) + 2\left(\sum_{j=n}^{n+M-1} \quad 2^j\left(\frac{I-Z_j}{2}\right) \times \sum_{i=0}^{n-1} \quad W_i\left(\frac{I-Z_i}{2}\right)\right)\Bigg) \\
&-\sum_{i=0}^{n-1} \quad P_i x_i
\end{aligned}
\tag{11}
$$

Hay que tener en cuenta que no es necesario calcular los tensores por cuenta propia, ya que Qiskit tiene incorporado un objeto que toma los índices de la matriz de Pauli Z en el tensor y realiza los cálculos restantes. Como resultado se obtiene una lista de objetos de Pauli y un valor shift que se encarga se acumular la diferencia de las magnitudes de la operación de cada sumatoria. Esos valores se envían al algoritmo ExactEigensolver con el objetivo de minimizar la función y retornar una solución X' donde los primeros n términos son la solución al problema de la mochila.

```
#term for sum(x_i*w_i)^2
for i in range(n):
    for j in range(n):
        coef = -1 * 0.25 * weights[i] * weights[j] * M

        xp = np.zeros(num_values, dtype=np.bool)
        zp = np.zeros(num_values, dtype=np.bool)
        zp[j] = not zp[j]
        pauli_list.append([coef, Pauli(zp, xp)])
        shift -= coef

        xp = np.zeros(num_values, dtype=np.bool)
        zp = np.zeros(num_values, dtype=np.bool)
        zp[i] = not zp[i]
        pauli_list.append([coef, Pauli(zp, xp)])
        shift -= coef

        coef = -1 * coef
        xp = np.zeros(num_values, dtype=np.bool)
        zp = np.zeros(num_values, dtype=np.bool)
        zp[i] = not zp[i]
        zp[j] = not zp[j]
        pauli_list.append([coef, Pauli(zp, xp)])
        shift -= coef
```

Figura 8. Operadores de Pauli para el término $(\sum_{i=0}^{n-1} \quad W_i x_i)^2$

4.2 TENDENCIAS EN APLICACIONES Y ALGORITMOS CUÁNTICOS QA&A EN QISKIT

Durante la fase de ejecución de este trabajo se presentó un imprevisto, las directrices de IBM deciden cambiar la estructura organizacional de los componentes que hacen parte de Quiskit teniendo en mente que “tener algoritmos cuánticos que se ejecutan rápidamente simplemente no es suficiente para ser convincente en este momento, porque ninguno de estos algoritmos es útil para nadie, y probablemente estén muy lejos de los algoritmos que finalmente proporcionarán ventajas cuánticas. Los usuarios se preocupan por construir y experimentar rápidamente con nuevos algoritmos, y podría decirse que una biblioteca con bloques de construcción de algorítmicos extremadamente sólidos, incluso uno sin algoritmos de caja negra en absoluto, puede ser más convincente” [47]. Es por esto por lo que, se decide ir en una nueva dirección en el software QA&A y se realizan tres modificaciones importantes en el conjunto de herramientas de aplicaciones y algoritmos cuánticos de Qiskit.

En primer lugar, se introduce una librería de circuitos en Qiskit Terra, proporcionando familias de circuitos con características mejoradas que son interesantes por sus aplicaciones prácticas y/o propiedades teóricas complejas.

En segundo lugar, se reconstruyen las herramientas algorítmicas centrales de Qiskit para optimizarlas para la investigación y la creación de prototipos, incluido un sistema completamente nuevo para construir flujos computacionales cuánticos, una reorganización completa de la librería, la introducción de algoritmos jerárquicos y un soporte más profundo para circuitos cuánticos.

En tercer lugar, se presenta un nuevo módulo de optimización, una librería para investigadores y principiantes por igual para participar en el desarrollo y la experimentación en la optimización combinatoria cuántica.

La introducción de este nuevo módulo de optimización trajo consigo que la librería Qiskit Aqua, sobre la cual se contempló desarrollar este trabajo inicialmente, quede obsoleta. Así, el trabajo que se realizó previamente en [18], del cual parte la investigación, queda como base de estudio para entender el funcionamiento interno del módulo de optimización, ya que se continúa utilizando el modelo de Ising para calcular el estado fundamental de un Hamiltoniano pero la librería ya cuenta con clases genéricas que dan soporte a los cálculos de los operadores de Pauli descritos en la **Ecuación 11.**

4.3 DESCRIPCIÓN DEL ALGORITMO Y SUS COMPONENTES

El problema de la mochila requiere hallar una combinación de artículos tal que el peso total esté dentro de la capacidad de la mochila y maximice el valor total de los artículos como se definió en la **Ecuación (1)**. Luego de obtener la **Ecuación (10)**, es fácil mapear el problema en una computadora cuántica, y la solución se encontrará minimizando los valores propios de un hamiltoniano de Ising. A continuación, se describen los componentes del algoritmo que da solución al problema de la mochila haciendo uso de Numpy eigensolver.

4.3.1 Librerías requeridas

Para la implementación del algoritmo que permite dar solución al problema de la mochila con el soporte de la librería Qiskit se deben importar los componentes indicados en la **Figura 9.** Librerías de Qiskit necesarias.

```
# Qiskit libraries
from qiskit import Aer
from qiskit.utils import algorithm_globals, QuantumInstance
from qiskit.algorithms import QAOA
from qiskit.algorithms import NumPyMinimumEigensolver
from qiskit_optimization.applications import Knapsack
from qiskit_optimization.algorithms import MinimumEigenOptimizer
```

Figura 9. Librerías de Qiskit necesarias

4.3.2 Mapeo a un problema de Ising

Para mapear el problema de la mochila binaria a un problema de Ising se hace necesario instanciar un objeto de la clase **Knapsack** del módulo de aplicaciones de la librería *qiskit_optimization,* y posteriormente se debe cambiar su representación a un programa cuadrático, que admite restricciones de desigualdad e igualdad y

variables continuas, binarias y enteras. La instanciación de variables se presenta en la **Figura 10**.

```
prob = Knapsack(
    values=[x0, x1, ... xn-1],
    weights=[W0, W1, ... Wn-1],
    max_weight=C
)
qp = prob.to_quadratic_program()
```

Figura 10. Transformación de una instancia *Knapsack* a un programa cuadrático

Así, en la variable **qp** se modifica la representación de un problema de mochila binaria a un problema de Ising donde su solución se encuentra minimizando los valores propios de su hamiltoniano.

4.3.3 Encontrar el estado base de un Hamiltoniano

Dado que la solución al problema de optimización original se obtiene encontrando el estado base de un Hamiltoniano, se requiere el uso de un algoritmo que permita optimizar los vectores propios mínimos del problema de Ising detallado en la **sección 4.1.2**, para ello en la librería *qiskit_optimization* se dispone del algoritmo *MinimunEigenOptimizer()* que recibe como parámetro un optimizador de estados de hamiltonianos y permite encontrar su solución ejecutando el método solve() el cual recibe nuestro problema de Ising *qp* como parámetro de entrada; en la **Figura 11** se detalla este procedimiento.

```
# Numpy Eigensolver
meo = MinimumEigenOptimizer(
    min_eigen_solver=NumPyMinimumEigensolver()
)
result = meo.solve(qp)
solution = result.x.astype(int).tolist()
```

Figura 11. Secuencia de ejecución del algoritmo *MinimunEigenOptimizer*

De esta manera en la variable **solution** se encuentra una solución probable a un problema de la mochila binaria.

En un contexto general, es posible reproducir este procedimiento para solucionar cualquier tipo de problema de optimización, y en caso de no existir las clases apropiadas o necesarias para el algoritmo, el módulo de optimización sirve como un conjunto de herramientas nativo para los investigadores. Es interoperable con su software estándar, que sirve como punto de entrada y traducción guiada a las

estructuras más profundas de Qiskit que permitan desarrollar soluciones específicas y personalizadas en la medida de los conocimientos y requerimientos del usuario.

CAPÍTULO 5

5 EXPERIMENTOS Y RESULTADOS

Dado que la cantidad de qubits necesarios para solucionar el problema de la mochila mediante lógica cuántica es proporcional al peso de la mochila y a la cantidad de elementos, actualmente se hace imposible tratar problemas con más de 15 elementos en entornos de simulación cuántica sobre computadores personales convencionales; Esto se debe a que un sistema con n-qubits puede representar 2^n estados al mismo tiempo, lo cual se convierte en una tarea difícil de realizar con la actual capacidad de procesamiento en estas computadores.

Se utilizaron diferentes conjuntos de datos combinando tres tipos de correlación con tres niveles de dificultad y un conjunto de rangos que determinan la cantidad de artículos que va a contener la mochila y el rango que sirve como valor máximo para la generación de los pesos de cada artículo de la mochila. Las pruebas se realizaron sobre un equipo personal con 12 Gigabytes de memoria RAM, un procesador Intel(R) Core (TM) i7-10510U CPU @ 1.80 GHz y un sistema operativo Windows 10 edición Profesional de 64-bits.

5.1 PARÁMETROS DE CONFIGURACIÓN

Esta subsección presenta los valores de los parámetros de ejecución de los algoritmos. Cabe mencionar que todos los experimentos referentes a los algoritmos seleccionados se realizaron en el mismo sistema, con una población de 20 individuos y un máximo de 1000 iteraciones. Los otros valores de parámetros son los valores predeterminados. Los ajustes generales de los parámetros están expuestos en la **Tabla 1**.

Tabla 1. Parámetros de configuración para los tres algoritmos

Parámetros	**Valor**
Tamaño de la población	20
Máximo de iteraciones (EFOS)	1000
Número de ejecuciones	31

5.2 CONJUNTO DE DATOS

Teniendo presente que las pruebas se van a realizar con instancias de baja dimensionalidad y que el módulo generador permite seleccionar diferentes configuraciones, la evaluación de los algoritmos se realizó con 9 conjuntos de datos de 30 problemas cada uno, discriminados por el tipo de correlación [1, 2, 3], el nivel de dificultad (fácil, media, difícil), con rangos de 5 a 10 con incrementos de +1 para

el número de instancias y de 10 a 50 con incrementos de +10 para el rango. En la **Tabla 2** se presentan los parámetros para la generación y ejecución de las diferentes configuraciones para cada conjunto de datos.

Tabla 2. Parámetros de ejecución para la generación de los problemas

Conjunto de datos	Parámetros de ejecución
1	*python* main.py -i 31 generate -t 1 -d 1 -n 5 10 -r 10 50
2	*python* main.py -i 31 generate -t 1 -d 2 -n 5 10 -r 10 50
3	*python* main.py -i 31 generate -t 1 -d 3 -n 5 10 -r 10 50
4	*python* main.py -i 31 generate -t 2 -d 1 -n 5 10 -r 10 50
5	*python* main.py -i 31 generate -t 2 -d 2 -n 5 10 -r 10 50
6	*python* main.py -i 31 generate -t 2 -d 3 -n 5 10 -r 10 50
7	*python* main.py -i 31 generate -t 3 -d 1 -n 5 10 -r 10 50
8	*python* main.py -i 31 generate -t 3 -d 2 -n 5 10 -r 10 50
9	*python* main.py -i 31 generate -t 3 -d 3 -n 5 10 -r 10 50

Cada conjunto de datos se almacena en una carpeta individual que a su vez contiene una carpeta con el nombre del artefacto ejecutable, y dentro de esta se encuentran 30 problemas de mochila que resultan de la combinación de los parámetros ingresados para cada conjunto de datos

Cabe señalar que los criterios estadísticos de tasa de éxito, promedio de los fitness obtenidos en las 31 iteraciones, la desviación estándar (SD) de los valores de fitness encontrados, mejor fitness obtenido, peor fitness obtenido y el promedio de los tiempos de ejecución obtenidos, se utilizaron para evaluar el desempeño de los algoritmos.

5.3 ALGORITMOS COMPARADOS

Para realizar el proceso de comparación del algoritmo cuántico implementado con base en la librería de Qiskit se seleccionaron tres algoritmos que basan su funcionamiento en metaheurísticas poblacionales, estos algoritmos son:

- An enhanced binary slime mould algorithm (SMA) for solving the 0–1 knapsack problem, publicado en 2021 en la revista JCR titulada Engineering with Computers (ISSN: 0177-0667) de Springer con factor de impacto para 2020 de 7.963 y cuartil Q1 en el puesto 8 de 111 en el tema de aplicaciones interdisciplinarias (Ciencias de la Computación) y en el 4 de 133 en el tema de Ingeniería Mecánica.
- Quantum-Inspired Differential Evolution with Grey Wolf Optimizer for 0-1 Knapsack Problem (QDGWO) publicado en 2021 en la revista JCR titulada Mathematics (eISSN: 2227-7390) de MDPI con factor de impacto para 2020 de 2.258 y cuartil Q1 en el puesto 24 de 330 en el tema de matemáticas.

- A Hybridization of Dragonfly Algorithm Optimization and Angle Modulation Mechanism for 0-1 Knapsack Problems (DA+IAMDA) publicado en 2021 en la revista JCR titulada Entropy (eISSN: 1099-4300) de MDPI con factor de impacto para 2020 de 2.524 y cuartil Q2 en el puesto 38 de 86 en el tema de Física (multidisciplinaria).

Con esta selección se cubren algoritmos que incorporan mecanismos de solución novedosos (publicados en 2021) o incorporan algún componente de computación cuántica, algoritmos que además son publicados en revistas JCR de alto impacto (Q1 y Q2) con lo cual se tiene mayor garantía de la calidad de estos. Los parámetros de configuración específicos de cada algoritmo se presentan en la **Tabla 3.**

Tabla 3. Parámetros de configuración específicos para cada algoritmo

Algoritmo	**Parámetros**
SMA	_EPSILON = 10E-10 _MU = 1.0 _SIGMA = 0.5
QDGWO	ELITISM_WOLVES = 3 F0 = 0.02 F1 = 0.03 THETA_MIN = 0.03141 (0.01*pi) THETA_MAX = 0.09424 (0.03*pi) _MU = 0 _SIGMA = 1 K = 10
DA+IAMDA	DIM_DA = 5 LB_DA = -1 UB_DA = 1

5.4 PROTOCOLO DE EXPERIMENTACIÓN

La fase de experimentación se realizó en nueve etapas de ejecución. Cada etapa inicia con la puesta en marcha de los algoritmos tomando los parámetros de ejecución indicados en la **Tabla 3**. Estos algoritmos fueron implementados sobre el marco de trabajo de prueba propuesto tomando la lógica de programación que cada autor detalla en los pseudocódigos de los artículos seleccionados.

Al momento de la ejecución, el metodo main() genera el conjunto de datos y se almacena en el fichero **files** del directorio del proyecto que se encuentra en el **Anexo A**, posteriormente el algoritmo lee los archivos generados, los almacena en memoria y uno a uno son enviados a cada metaheurística para realizar el proceso de encontrar la mejor solución posible al conjunto de datos (problema o instancia de la mochila binaria).

Para realizar la evaluación y las comparaciones entre los diferentes escenarios de evaluación obtenidos con base en la diversidad del conjunto de datos generado, se calcula la tasa de éxito, se realizaron mediciones del promedio de fitness, del máximo (mejor) fitness encontrado, mínimo (peor) fitness encontrado y se calcula el

tiempo promedio de ejecución de cada algoritmo; Esto con el objetivo de evaluar tanto la calidad de las soluciones como la rapidez de convergencia de cada algoritmo. Estas mediciones se realizaron tomando el resultado promedio de 31 ejecuciones y los resultados son almacenados en los archivos mencionados en la **sección 3.2.3.2.**

Finalmente, con los archivos que detallan los resultados de las ejecuciones se procede a realizar un análisis estadístico de los datos aplicando un test de Friedman con la herramienta **KEEL suite 3.0** (http://keel.es/) para analizar el comportamiento de cada algoritmo.

5.5 RESULTADOS EXPERIMENTALES

A continuación, se presentan los resultados obtenidos de realizar las nueve etapas de ejecución de los algoritmos seleccionados con sus correspondientes conjuntos de datos. Cabe mencionar que en las tablas presentadas se consignan los promedios de los fitness encontrados y los tiempos de ejecución resultantes de las 31 ejecuciones realizadas por cada archivo.

Debido a la gran cantidad de datos de datos obtenidos, los resultados se van a presentar agrupados por el tipo de correlación que existe entre los artículos (elementos u objetos) de la mochila.

5.5.1 Resultados para instancias no correlacionadas

En las tablas presentadas a continuación se presentan los resultados correspondientes a la ejecución de las pruebas para instancias no correlacionadas (parámetro t = 1).

5.5.1.1 Promedio de fitness encontrados

Tabla 4. Promedio de fitness para la configuración t1_d1_n5-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t1_d1_n5_r10.dat	19	17	19	19
t1_d1_n5_r20.dat	19	19	19	19
t1_d1_n5_r30.dat	76	76	76	76
t1_d1_n5_r40.dat	59	59	59	59
t1_d1_n5_r50.dat	81	81	81	81
t1_d1_n6_r10.dat	23	23	23	23
t1_d1_n6_r20.dat	23	23	23	23
t1_d1_n6_r30.dat	76	76	75.483871	76
t1_d1_n6_r40.dat	59	59	59	59
t1_d1_n6_r50.dat	81	81	81	81
t1_d1_n7_r10.dat	23	23	22.9354839	23
t1_d1_n7_r20.dat	23	23	23	23

t1_d1_n7_r30.dat	76	76	73.8709677	76
t1_d1_n7_r40.dat	59	59	58.483871	59
t1_d1_n7_r50.dat	106	106	105.516129	106
t1_d1_n8_r10.dat	23	23	22.7419355	23
t1_d1_n8_r20.dat	23	23	23	23
t1_d1_n8_r30.dat	77	77	76.8387097	77
t1_d1_n8_r40.dat	61	61	60.9354839	61
t1_d1_n8_r50.dat	106	106	104.548387	106
t1_d1_n9_r10.dat	25	23	24.3870968	25
t1_d1_n9_r20.dat	38	38	37.5483871	38
t1_d1_n9_r30.dat	77	77	76.3225806	77
t1_d1_n9_r40.dat	61	61	60.7741935	61
t1_d1_n9_r50.dat	106	106	105.193548	106
t1_d1_n10_r10.dat	25	23	24.2258065	25
t1_d1_n10_r20.dat	38	38	37.5483871	38
t1_d1_n10_r30.dat	77	77	76.7419355	77
t1_d1_n10_r40.dat	81	81	79.1935484	81
t1_d1_n10_r50.dat	106	106	103.774194	106

Tabla 5. Promedio de fitness para la configuración t1_d2_n5-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t1_d2_n5_r10.dat	23	16	22.7741935	23
t1_d2_n5_r20.dat	36	36	36	36
t1_d2_n5_r30.dat	63	63	63	63
t1_d2_n5_r40.dat	76	76	76	76
t1_d2_n5_r50.dat	113	113	113	113
t1_d2_n6_r10.dat	23	18	22.6774194	23
t1_d2_n6_r20.dat	36	36	36	36
t1_d2_n6_r30.dat	63	63	63	63
t1_d2_n6_r40.dat	125	125	125	125
t1_d2_n6_r50.dat	113	113	113	113
t1_d2_n7_r10.dat	28	26	27.9354839	28
t1_d2_n7_r20.dat	46	46	46	46
t1_d2_n7_r30.dat	66	66	65.7096774	66
t1_d2_n7_r40.dat	135	135	134.032258	135
t1_d2_n7_r50.dat	113	113	113	113
t1_d2_n8_r10.dat	28	26	27.8064516	28
t1_d2_n8_r20.dat	51	46	50.8387097	51
t1_d2_n8_r30.dat	66	66	66	66

t1_d2_n8_r40.dat	138	138	138	138
t1_d2_n8_r50.dat	113	113	111.903226	113
t1_d2_n9_r10.dat	28	28	27.7419355	28
t1_d2_n9_r20.dat	58	58	57.3870968	58
t1_d2_n9_r30.dat	66	66	65.2258065	66
t1_d2_n9_r40.dat	138	138	137.903226	138
t1_d2_n9_r50.dat	113	113	112.516129	113
t1_d2_n10_r10.dat	30	30	29.3548387	30
t1_d2_n10_r20.dat	72	62	69.0645161	72
t1_d2_n10_r30.dat	66	66	65.516129	66
t1_d2_n10_r40.dat	138	138	136.322581	138
t1_d2_n10_r50.dat	117	117	114.903226	117

Tabla 6. Promedio de fitness para la configuración t1_d3_n5-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t1_d3_n5_r10.dat	15	15	14.9032258	15
t1_d3_n5_r20.dat	19	19	19	19
t1_d3_n5_r30.dat	53	53	53	53
t1_d3_n5_r40.dat	43	43	43	43
t1_d3_n5_r50.dat	72	58	72	72
t1_d3_n6_r10.dat	23	20	23	23
t1_d3_n6_r20.dat	23	23	23	23
t1_d3_n6_r30.dat	53	53	53	53
t1_d3_n6_r40.dat	43	43	43	43
t1_d3_n6_r50.dat	72	72	71.516129	72
t1_d3_n7_r10.dat	23	23	23	23
t1_d3_n7_r20.dat	23	23	23	23
t1_d3_n7_r30.dat	53	53	53	53
t1_d3_n7_r40.dat	59	59	58.483871	59
t1_d3_n7_r50.dat	92	81	91.8064516	92
t1_d3_n8_r10.dat	23	23	22.4193548	23
t1_d3_n8_r20.dat	23	23	23	23
t1_d3_n8_r30.dat	53	53	53	53
t1_d3_n8_r40.dat	61	61	60.9354839	61
t1_d3_n8_r50.dat	92	81	91.4193548	92
t1_d3_n9_r10.dat	23	23	22.2258065	23
t1_d3_n9_r20.dat	38	38	37.5483871	38
t1_d3_n9_r30.dat	77	77	76.8709677	77
t1_d3_n9_r40.dat	61	61	60.7741935	61

t1_d3_n9_r50.dat	92	81	91.0967742	92
t1_d3_n10_r10.dat	23	23	21.6129032	23
t1_d3_n10_r20.dat	38	38	37.7096774	38
t1_d3_n10_r30.dat	77	77	76.8064516	77
t1_d3_n10_r40.dat	81	81	79.2580645	81
t1_d3_n10_r50.dat	92	81	91	92

5.5.1.2 Promedio de tiempos de ejecución encontrados

Tabla 7. Promedio de tiempos para la configuración t1_d1_n6-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t1_d1_n5_r10.dat	0.41	292.05	514.45	5.33
t1_d1_n5_r20.dat	0.43	181.58	496.51	7.05
t1_d1_n5_r30.dat	0.48	288.68	494.15	7.01
t1_d1_n5_r40.dat	0.41	181.23	503.37	10.69
t1_d1_n5_r50.dat	0.44	187.18	496.7	10.74
t1_d1_n6_r10.dat	0.47	475.58	498.3	6.94
t1_d1_n6_r20.dat	0.44	199.77	515.65	10.67
t1_d1_n6_r30.dat	0.46	223.71	501.12	10.95
t1_d1_n6_r40.dat	0.46	201.47	507.94	19.36
t1_d1_n6_r50.dat	0.49	211.28	500.01	18.63
t1_d1_n7_r10.dat	0.49	227.9	501.85	10.77
t1_d1_n7_r20.dat	0.51	221.05	504.36	18.6
t1_d1_n7_r30.dat	0.6	230.27	500.93	18.44
t1_d1_n7_r40.dat	0.46	220.13	511.87	35.73
t1_d1_n7_r50.dat	0.52	231.02	504.66	35.69
t1_d1_n8_r10.dat	0.59	244.21	515.03	19.15
t1_d1_n8_r20.dat	0.61	484.23	741.23	42.01
t1_d1_n8_r30.dat	0.64	268.97	504.49	36.78
t1_d1_n8_r40.dat	0.59	247.53	505.75	74.66
t1_d1_n8_r50.dat	0.6	256.74	506.31	74.81
t1_d1_n9_r10.dat	0.77	267.33	505.43	36.11
t1_d1_n9_r20.dat	0.65	266.26	502.83	74.5
t1_d1_n9_r30.dat	0.68	269.95	512.24	73.87
t1_d1_n9_r40.dat	0.64	264.21	499.86	160.91
t1_d1_n9_r50.dat	0.64	265.73	498.61	160.81
t1_d1_n10_r10.dat	0.81	258.54	497.14	73.98
t1_d1_n10_r20.dat	0.67	284.44	502.09	160.25
t1_d1_n10_r30.dat	0.75	286.92	521.64	160.43
t1_d1_n10_r40.dat	0.7	280.92	504.4	361.66

t1_d1_n10_r50.dat	0.69	284.15	505.76	363.29

Tabla 8. Promedio de tiempos para la configuración t1_d2_n6-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t1_d2_n5_r10.dat	0.4	187.23	506.77	5.38
t1_d2_n5_r20.dat	0.39	187.74	510.18	7.63
t1_d2_n5_r30.dat	0.4	196.42	520.11	7.58
t1_d2_n5_r40.dat	0.43	188.77	519.25	7.38
t1_d2_n5_r50.dat	0.37	194.26	515.09	11.65
t1_d2_n6_r10.dat	0.45	222.73	522.54	7.84
t1_d2_n6_r20.dat	0.47	214.02	523.31	12.08
t1_d2_n6_r30.dat	0.45	217.75	519.56	12.25
t1_d2_n6_r40.dat	0.53	228.8	531	20.94
t1_d2_n6_r50.dat	0.51	219.47	523.65	20.21
t1_d2_n7_r10.dat	0.56	245.46	521.3	11.61
t1_d2_n7_r20.dat	0.53	243.66	544.19	20.83
t1_d2_n7_r30.dat	0.53	255.61	546.94	20.85
t1_d2_n7_r40.dat	0.59	266.16	550.06	40.26
t1_d2_n7_r50.dat	0.61	253.2	547.95	41.64
t1_d2_n8_r10.dat	0.68	276.93	713.58	27.29
t1_d2_n8_r20.dat	0.86	357.73	699.13	51.1
t1_d2_n8_r30.dat	0.75	318.15	554.1	38.74
t1_d2_n8_r40.dat	0.67	276.13	541.52	80.9
t1_d2_n8_r50.dat	0.65	263.15	549.51	80.39
t1_d2_n9_r10.dat	0.92	285.82	547.11	40.05
t1_d2_n9_r20.dat	0.77	285.75	549.28	81
t1_d2_n9_r30.dat	0.73	284.79	550.51	80.45
t1_d2_n9_r40.dat	0.77	300.59	554.51	177.68
t1_d2_n9_r50.dat	0.77	300.1	560.61	180.7
t1_d2_n10_r10.dat	0.83	308.71	546.52	80.33
t1_d2_n10_r20.dat	0.72	299.88	549.08	176.88
t1_d2_n10_r30.dat	0.75	307.71	548.92	177.3
t1_d2_n10_r40.dat	0.81	317.28	551.42	395.63
t1_d2_n10_r50.dat	1	300.99	570.49	409.53

Tabla 9. Promedio de tiempos para la configuración t1_d3_n6-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t1_d3_n5_r10.dat	0.51	200.3	544.91	5.59
t1_d3_n5_r20.dat	0.4	196.41	547.64	8.09
t1_d3_n5_r30.dat	0.39	195.57	545.67	5.83

t1_d3_n5_r40.dat	0.46	196.05	548.45	12.33
t1_d3_n5_r50.dat	0.42	200.26	548.93	12.72
t1_d3_n6_r10.dat	0.44	229.63	563.91	8.37
t1_d3_n6_r20.dat	0.49	219.33	544.6	12.06
t1_d3_n6_r30.dat	1.39	215.22	488.91	11.01
t1_d3_n6_r40.dat	0.49	198.54	815.03	21.12
t1_d3_n6_r50.dat	0.59	221.77	894.07	55
t1_d3_n7_r10.dat	0.54	236.58	537.58	11.68
t1_d3_n7_r20.dat	0.54	237.75	525.75	20.44
t1_d3_n7_r30.dat	1.72	643.92	764.38	25.08
t1_d3_n7_r40.dat	0.73	286.58	1288.67	119.46
t1_d3_n7_r50.dat	1.75	549.08	649.67	47.69
t1_d3_n8_r10.dat	0.67	289.16	608.7	23.26
t1_d3_n8_r20.dat	0.69	291.51	592.76	42.85
t1_d3_n8_r30.dat	0.67	293.67	982.98	42.55
t1_d3_n8_r40.dat	0.68	282.48	883.39	89.21
t1_d3_n8_r50.dat	0.81	288.61	599.35	89.42
t1_d3_n9_r10.dat	0.7	316.85	1070.37	103.34
t1_d3_n9_r20.dat	1.75	409.57	594.38	91.72
t1_d3_n9_r30.dat	0.8	313.03	598.85	88.35
t1_d3_n9_r40.dat	0.81	314.76	601.86	196.26
t1_d3_n9_r50.dat	0.71	328.3	601.28	196.03
t1_d3_n10_r10.dat	0.85	299.58	549.66	82.49
t1_d3_n10_r20.dat	0.73	307.87	561.64	195.94
t1_d3_n10_r30.dat	0.79	327.28	595.27	193.04
t1_d3_n10_r40.dat	0.9	327.82	595.6	440.25
t1_d3_n10_r50.dat	0.83	344.13	599.29	437.03

5.5.2 Resultados para instancias débilmente correlacionadas

En las tablas presentadas a continuación se presentan los resultados correspondientes a la ejecución de las pruebas para instancias débilmente correlacionadas (parámetro t = 2).

5.5.2.1 Promedio de fitness encontrados

Tabla 10. Promedio de fitness para la configuración t2_d1_n5-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t2_d1_n5_r10.dat	11	10	11	11
t2_d1_n5_r20.dat	22	22	22	22
t2_d1_n5_r30.dat	28	28	28	28
t2_d1_n5_r40.dat	40	40	40	40

t2_d1_n5_r50.dat	47	47	47	47
t2_d1_n6_r10.dat	11	10	11	11
t2_d1_n6_r20.dat	22	22	22	22
t2_d1_n6_r30.dat	30	30	29.9354839	30
t2_d1_n6_r40.dat	40	40	40	40
t2_d1_n6_r50.dat	47	47	47	47
t2_d1_n7_r10.dat	11	10	11	11
t2_d1_n7_r20.dat	24	22	24	24
t2_d1_n7_r30.dat	30	30	29.9354839	30
t2_d1_n7_r40.dat	40	40	40	40
t2_d1_n7_r50.dat	50	50	49.4193548	50
t2_d1_n8_r10.dat	12	11	12	12
t2_d1_n8_r20.dat	24	22	24	24
t2_d1_n8_r30.dat	30	30	29.8064516	30
t2_d1_n8_r40.dat	41	41	40.9032258	41
t2_d1_n8_r50.dat	50	50	49.7741935	50
t2_d1_n9_r10.dat	12	11	12	12
t2_d1_n9_r20.dat	25	25	24.9354839	25
t2_d1_n9_r30.dat	33	33	33	33
t2_d1_n9_r40.dat	41	41	40.7741935	41
t2_d1_n9_r50.dat	52	52	51.8064516	52
t2_d1_n10_r10.dat	12	11	12	12
t2_d1_n10_r20.dat	25	25	24.7096774	25
t2_d1_n10_r30.dat	33	33	32.5806452	33
t2_d1_n10_r40.dat	46	44	44.8709677	46
t2_d1_n10_r50.dat	57	57	56.3870968	57

Tabla 11. Promedio de fitness para la configuración t2_d2_n5-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t2_d2_n5_r10.dat	12	10	11.9677419	12
t2_d2_n5_r20.dat	15	14	15	15
t2_d2_n5_r30.dat	35	33	34.9354839	35
t2_d2_n5_r40.dat	15	15	15	15
t2_d2_n5_r50.dat	49	42	49	49
t2_d2_n6_r10.dat	12	10	11.9032258	12
t2_d2_n6_r20.dat	15	15	15	15
t2_d2_n6_r30.dat	35	33	34.8709677	35
t2_d2_n6_r40.dat	40	39	40	40
t2_d2_n6_r50.dat	49	47	49	49
t2_d2_n7_r10.dat	13	12	13	13

t2_d2_n7_r20.dat	18	16	17.9677419	18
t2_d2_n7_r30.dat	35	33	34.7419355	35
t2_d2_n7_r40.dat	40	40	39.9677419	40
t2_d2_n7_r50.dat	50	47	50	50
t2_d2_n8_r10.dat	13	12	12.7419355	13
t2_d2_n8_r20.dat	20	18	19.3870968	20
t2_d2_n8_r30.dat	35	33	34.7419355	35
t2_d2_n8_r40.dat	40	40	39.8709677	40
t2_d2_n8_r50.dat	50	47	49.9032258	50
t2_d2_n9_r10.dat	13	12	12.9032258	13
t2_d2_n9_r20.dat	20	18	19.2258065	20
t2_d2_n9_r30.dat	35	33	33.9354839	35
t2_d2_n9_r40.dat	42	41	41.7419355	42
t2_d2_n9_r50.dat	50	47	49.9032258	50
t2_d2_n10_r10.dat	13	12	12.8064516	13
t2_d2_n10_r20.dat	20	20	19.7096774	20
t2_d2_n10_r30.dat	35	33	34.3548387	35
t2_d2_n10_r40.dat	42	41	41.7741935	42
t2_d2_n10_r50.dat	50	47	49.8064516	50

Tabla 12. Promedio de fitness para la configuración t2_d3_n5-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t2_d3_n5_r10.dat	8	8	8	8
t2_d3_n5_r20.dat	19	19	19	19
t2_d3_n5_r30.dat	13	13	13	13
t2_d3_n5_r40.dat	35	35	34.9677419	35
t2_d3_n5_r50.dat	36	32	36	36
t2_d3_n6_r10.dat	9	8	9	9
t2_d3_n6_r20.dat	19	19	19	19
t2_d3_n6_r30.dat	23	18	23	23
t2_d3_n6_r40.dat	35	35	35	35
t2_d3_n6_r50.dat	40	40	40	40
t2_d3_n7_r10.dat	10	10	10	10
t2_d3_n7_r20.dat	22	22	22	22
t2_d3_n7_r30.dat	23	19	23	23
t2_d3_n7_r40.dat	40	40	40	40
t2_d3_n7_r50.dat	42	40	41.7096774	42
t2_d3_n8_r10.dat	11	10	10.9677419	11
t2_d3_n8_r20.dat	22	22	22	22
t2_d3_n8_r30.dat	23	19	23	23

t2_d3_n8_r40.dat	41	41	40.9032258	41
t2_d3_n8_r50.dat	42	41	41.8064516	42
t2_d3_n9_r10.dat	11	10	10.9677419	11
t2_d3_n9_r20.dat	25	22	24.2258065	25
t2_d3_n9_r30.dat	27	23	26.8709677	27
t2_d3_n9_r40.dat	41	41	40.7741935	41
t2_d3_n9_r50.dat	46	41	45.6129032	46
t2_d3_n10_r10.dat	11	10	11	11
t2_d3_n10_r20.dat	25	22	24.6774194	25
t2_d3_n10_r30.dat	27	23	26.8709677	27
t2_d3_n10_r40.dat	44	44	43.1935484	44
t2_d3_n10_r50.dat	46	43	45.7096774	46

5.5.2.2 Promedio de tiempos de ejecución encontrados

Tabla 13. Promedio de tiempos para la configuración t2_d1_n5-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t2_d1_n5_r10.dat	0.39	206.59	522.51	5.58
t2_d1_n5_r20.dat	0.51	189.19	521.58	7.68
t2_d1_n5_r30.dat	0.51	328.84	523.68	7.73
t2_d1_n5_r40.dat	0.52	196.68	520.47	11.65
t2_d1_n5_r50.dat	0.49	208.74	523.65	11.82
t2_d1_n6_r10.dat	0.54	227.6	525.75	7.75
t2_d1_n6_r20.dat	0.54	211.82	523.46	11.77
t2_d1_n6_r30.dat	0.61	223.87	522.38	11.89
t2_d1_n6_r40.dat	0.55	215.43	529.96	20.4
t2_d1_n6_r50.dat	0.56	219.3	524.6	20.31
t2_d1_n7_r10.dat	0.62	246.59	525.96	11.61
t2_d1_n7_r20.dat	0.62	233.8	524.93	20.32
t2_d1_n7_r30.dat	1.02	449.35	609.42	20.51
t2_d1_n7_r40.dat	0.63	237.08	530.99	38.7
t2_d1_n7_r50.dat	0.58	276.04	532.19	38.93
t2_d1_n8_r10.dat	0.72	261.64	525.46	20.31
t2_d1_n8_r20.dat	1.03	256.77	526.26	38.96
t2_d1_n8_r30.dat	1.29	273.52	526.32	38.68
t2_d1_n8_r40.dat	0.64	257.98	527.5	79.73
t2_d1_n8_r50.dat	0.65	270.73	525.68	80.76
t2_d1_n9_r10.dat	0.78	289.26	528.13	39.07
t2_d1_n9_r20.dat	0.75	286.66	530.23	81.55
t2_d1_n9_r30.dat	0.64	338.44	582.63	88.31
t2_d1_n9_r40.dat	0.78	301.95	540.83	184.41

t2_d1_n9_r50.dat	0.82	313.73	587.29	194.52
t2_d1_n10_r10.dat	1.82	629.33	729.33	88.97
t2_d1_n10_r20.dat	0.94	588.18	672.01	180.91
t2_d1_n10_r30.dat	0.8	320.42	536.22	177.07
t2_d1_n10_r40.dat	0.81	306.15	532.65	398.19
t2_d1_n10_r50.dat	0.83	312.1	536.45	395.74

Tabla 14. Promedio de tiempos para la configuración t2_d2_n5-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t2_d2_n5_r10.dat	0.54	201.99	551.42	5.75
t2_d2_n5_r20.dat	0.5	199.37	551.19	7.72
t2_d2_n5_r30.dat	0.47	203.73	552.49	8.09
t2_d2_n5_r40.dat	0.51	199.1	554.24	8.14
t2_d2_n5_r50.dat	0.47	203.91	545.95	12.23
t2_d2_n6_r10.dat	0.53	225.22	549.27	8.12
t2_d2_n6_r20.dat	0.45	223.08	554.62	12.62
t2_d2_n6_r30.dat	0.47	226.21	560.13	12.71
t2_d2_n6_r40.dat	0.68	233.67	558.47	21.54
t2_d2_n6_r50.dat	0.6	229.1	564.47	29.36
t2_d2_n7_r10.dat	0.69	347.42	1361.56	36.16
t2_d2_n7_r20.dat	1.63	728.72	841.1	25.83
t2_d2_n7_r30.dat	0.69	299	654.08	25.05
t2_d2_n7_r40.dat	0.67	309.95	654.65	49.01
t2_d2_n7_r50.dat	1.19	295.62	650.07	38.92
t2_d2_n8_r10.dat	0.65	318.96	634.28	23.19
t2_d2_n8_r20.dat	0.65	296.51	610.86	44.5
t2_d2_n8_r30.dat	0.64	298.95	612.14	44.65
t2_d2_n8_r40.dat	0.68	314.29	608.28	92.09
t2_d2_n8_r50.dat	1.56	297.98	609.21	92.24
t2_d2_n9_r10.dat	0.85	330.11	611.4	45.87
t2_d2_n9_r20.dat	0.73	324.23	603.58	92.79
t2_d2_n9_r30.dat	0.78	326.49	606.88	93.06
t2_d2_n9_r40.dat	1.04	338.44	623.65	219.16
t2_d2_n9_r50.dat	1.62	348.79	644.42	225.23
t2_d2_n10_r10.dat	0.84	330.03	557.37	85.31
t2_d2_n10_r20.dat	0.67	312.17	558.28	187.29
t2_d2_n10_r30.dat	0.73	324.39	559.93	183.74
t2_d2_n10_r40.dat	0.97	325.93	555.92	414.7
t2_d2_n10_r50.dat	3.23	332.72	560.13	420.37

Tabla 15. Promedio de tiempos para la configuración t2_d3_n6-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t2_d3_n5_r10.dat	0.46	248.9	693.71	7.46
t2_d3_n5_r20.dat	0.55	258.81	1046.03	8.14
t2_d3_n5_r30.dat	0.4	204.8	575.4	5.87
t2_d3_n5_r40.dat	0.49	353.16	1480.37	33.11
t2_d3_n5_r50.dat	0.62	525.08	1406.51	31.78
t2_d3_n6_r10.dat	1.32	314.52	1069.71	16.27
t2_d3_n6_r20.dat	1.16	423.92	1045.17	13.2
t2_d3_n6_r30.dat	0.56	232.22	554.05	11.6
t2_d3_n6_r40.dat	0.54	211.96	577	22.27
t2_d3_n6_r50.dat	0.52	233.92	579.55	22.29
t2_d3_n7_r10.dat	0.54	261.3	584.52	13.51
t2_d3_n7_r20.dat	0.71	258.55	536.42	20.68
t2_d3_n7_r30.dat	0.64	239.19	545.64	20.64
t2_d3_n7_r40.dat	0.67	242.2	539.74	41.65
t2_d3_n7_r50.dat	3.48	263.46	591.87	43
t2_d3_n8_r10.dat	0.79	282.57	578.15	21.88
t2_d3_n8_r20.dat	0.77	272.31	535.98	39.63
t2_d3_n8_r30.dat	0.71	263.31	537.66	39.61
t2_d3_n8_r40.dat	0.62	262.13	584.02	83.52
t2_d3_n8_r50.dat	1.67	262.03	545.13	88.56
t2_d3_n9_r10.dat	0.81	313.75	680.05	56.45
t2_d3_n9_r20.dat	0.76	278.51	520.04	79.2
t2_d3_n9_r30.dat	1.1	309.37	576.18	86.85
t2_d3_n9_r40.dat	0.77	307.81	582.71	190.91
t2_d3_n9_r50.dat	0.84	304.69	549.01	173.5
t2_d3_n10_r10.dat	1.02	399.43	695.29	109.75
t2_d3_n10_r20.dat	1.17	389.37	682.64	235.75
t2_d3_n10_r30.dat	1.1	396.55	687.46	231.55
t2_d3_n10_r40.dat	0.8	382.24	697.4	517.55
t2_d3_n10_r50.dat	1.21	406.86	694.87	518.15

5.5.3 Resultados para instancias fuertemente correlacionadas

En las tablas presentadas a continuación se resumen los resultados correspondientes a la ejecución de las pruebas para instancias fuertemente correlacionadas (parámetro t = 3). El conjunto de datos generado para este tipo de problemas tiene la particularidad que la cantidad de artículos (n) inicia en 6, esto debido a que el algoritmo DA+IAMDA no presentó respuesta de solución alguna después de tres horas de ejecución.

5.5.3.1 Promedio de fitness encontrados

Tabla 16. Promedio de fitness para la configuración t3_d1_n6-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t3_d1_n6_r10.dat	51	50	50.9677419	51
t3_d1_n6_r20.dat	60	60	59.8709677	60
t3_d1_n6_r30.dat	60	60	60	60
t3_d1_n6_r40.dat	69	69	69	69
t3_d1_n6_r50.dat	69	69	68.9032258	69
t3_d1_n7_r10.dat	51	50	50.9677419	51
t3_d1_n7_r20.dat	60	60	59.483871	60
t3_d1_n7_r30.dat	71	60	70.6451613	71
t3_d1_n7_r40.dat	79	79	78.0322581	79
t3_d1_n7_r50.dat	74	74	73.9677419	74
t3_d1_n8_r10.dat	51	50	50.9354839	51
t3_d1_n8_r20.dat	60	60	60	60
t3_d1_n8_r30.dat	71	60	70.6451613	71
t3_d1_n8_r40.dat	81	80	80.7096774	81
t3_d1_n8_r50.dat	89	89	86.0322581	89
t3_d1_n9_r10.dat	51	50	50.9354839	51
t3_d1_n9_r20.dat	60	60	59.5483871	60
t3_d1_n9_r30.dat	71	69	70.9354839	71
t3_d1_n9_r40.dat	81	80	80.8387097	81
t3_d1_n9_r50.dat	89	89	86.516129	89
t3_d1_n10_r10.dat	51	50	50.9354839	51
t3_d1_n10_r20.dat	60	60	59.2258065	60
t3_d1_n10_r30.dat	71	69	70.8387097	71
t3_d1_n10_r40.dat	81	80	80.1612903	81
t3_d1_n10_r50.dat	89	89	85.6451613	89

Tabla 17. Promedio de fitness para la configuración t3_d2_n6-10_r10-50

FileName	SlimeMould	GreyWolf	Dragonfly	QuantumEigensolver
t3_d2_n6_r10.dat	10	10	10	10
t3_d2_n6_r20.dat	22	22	22	22
t3_d2_n6_r30.dat	36	33	36	36
t3_d2_n6_r40.dat	47	47	47	47
t3_d2_n6_r50.dat	57	52	57	57
t3_d2_n7_r10.dat	12	10	11.9032258	12
t3_d2_n7_r20.dat	24	24	24	24
t3_d2_n7_r30.dat	36	36	35.9354839	36

t3_d2_n7_r40.dat	50	50	50	50
t3_d2_n7_r50.dat	57	52	56.8387097	57
t3_d2_n8_r10.dat	13	12	13	13
t3_d2_n8_r20.dat	25	25	25	25
t3_d2_n8_r30.dat	40	40	39.6451613	40
t3_d2_n8_r40.dat	51	50	51	51
t3_d2_n8_r50.dat	57	57	56.8387097	57
t3_d2_n9_r10.dat	13	12	12.9032258	13
t3_d2_n9_r20.dat	25	25	25	25
t3_d2_n9_r30.dat	40	40	39.3225806	40
t3_d2_n9_r40.dat	53	53	52.5483871	53
t3_d2_n9_r50.dat	57	57	56.3548387	57
t3_d2_n10_r10.dat	13	12	12.9354839	13
t3_d2_n10_r20.dat	25	25	25	25
t3_d2_n10_r30.dat	40	40	39.1290323	40
t3_d2_n10_r40.dat	53	53	52.1935484	53
t3_d2_n10_r50.dat	57	57	56.1935484	57

Tabla 18. Promedio de fitness para la configuración t3_d3_n6-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t3_d3_n6_r10.dat	10	9	10	10
t3_d3_n6_r20.dat	19	19	18.9032258	19
t3_d3_n6_r30.dat	32	28	32	32
t3_d3_n6_r40.dat	43	43	42.9032258	43
t3_d3_n6_r50.dat	56	56	56	56
t3_d3_n7_r10.dat	11	10	10.9677419	11
t3_d3_n7_r20.dat	24	24	24	24
t3_d3_n7_r30.dat	32	29	32	32
t3_d3_n7_r40.dat	44	44	43.9354839	44
t3_d3_n7_r50.dat	59	59	58.9677419	59
t3_d3_n8_r10.dat	13	11	12.7419355	13
t3_d3_n8_r20.dat	26	24	25.516129	26
t3_d3_n8_r30.dat	38	36	38	38
t3_d3_n8_r40.dat	46	46	45.9354839	46
t3_d3_n8_r50.dat	60	60	59.8709677	60
t3_d3_n9_r10.dat	13	11	12.5806452	13
t3_d3_n9_r20.dat	26	24	25.1935484	26
t3_d3_n9_r30.dat	41	40	40.516129	41
t3_d3_n9_r40.dat	50	46	49.1612903	50

t3_d3_n9_r50.dat	60	60	59.9032258	60
t3_d3_n10_r10.dat	13	11	12.7419355	13
t3_d3_n10_r20.dat	26	24	25.2903226	26
t3_d3_n10_r30.dat	41	40	40.8064516	41
t3_d3_n10_r40.dat	50	46	48.6129032	50
t3_d3_n10_r50.dat	62	62	61.9677419	62

5.5.3.2 Promedio de tiempo de ejecución

Tabla 19. Promedio de tiempos para la configuración t3_d1_n6-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t3_d1_n6_r10.dat	0.53	243.28	518.92	7.87
t3_d1_n6_r20.dat	0.56	292.03	521.36	11.67
t3_d1_n6_r30.dat	0.7	226.8	518.03	11.76
t3_d1_n6_r40.dat	0.6	229.46	524.51	20.38
t3_d1_n6_r50.dat	0.79	215.71	521.36	20.6
t3_d1_n7_r10.dat	0.58	254.36	649.34	18.1
t3_d1_n7_r20.dat	0.89	357.48	763.19	29.88
t3_d1_n7_r30.dat	0.79	348.47	696.32	21.49
t3_d1_n7_r40.dat	0.51	242.12	517.03	39.33
t3_d1_n7_r50.dat	0.69	239.67	525.84	40.12
t3_d1_n8_r10.dat	0.8	264.94	527.73	21.33
t3_d1_n8_r20.dat	0.7	267.88	521.71	39.51
t3_d1_n8_r30.dat	0.55	265.98	520.82	38.64
t3_d1_n8_r40.dat	0.77	269.1	522.95	80.16
t3_d1_n8_r50.dat	0.59	263.2	532.29	80.9
t3_d1_n9_r10.dat	0.73	291.6	530.92	39.22
t3_d1_n9_r20.dat	0.82	289.25	525.95	80.99
t3_d1_n9_r30.dat	0.75	286.26	524.74	80.59
t3_d1_n9_r40.dat	1.85	283.01	523.81	177.54
t3_d1_n9_r50.dat	0.67	281.26	523.79	175.94
t3_d1_n10_r10.dat	0.84	305	521.75	80.03
t3_d1_n10_r20.dat	0.8	305.67	527.56	175.62
t3_d1_n10_r30.dat	0.87	305.74	526.81	175.29
t3_d1_n10_r40.dat	1.36	295.99	526.19	394.39
t3_d1_n10_r50.dat	0.88	300.08	527.7	394.06

Tabla 20. Promedio de tiempos para la configuración t3_d2_n6-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t3_d2_n6_r10.dat	0.48	211.83	515.24	7.6
t3_d2_n6_r20.dat	0.65	210.89	516.3	11.47

t3_d2_n6_r30.dat	0.59	211.99	516.7	11.66
t3_d2_n6_r40.dat	0.53	214.3	519.48	19.94
t3_d2_n6_r50.dat	0.66	212.22	522.56	20.25
t3_d2_n7_r10.dat	0.81	233.58	522.28	11.65
t3_d2_n7_r20.dat	0.6	236.56	515.31	19.96
t3_d2_n7_r30.dat	0.63	251.69	1445.84	57.69
t3_d2_n7_r40.dat	1.4	664.99	1476.21	108.52
t3_d2_n7_r50.dat	3.12	661.34	1469.25	108.56
t3_d2_n8_r10.dat	1.76	719.7	1472.74	56.49
t3_d2_n8_r20.dat	1.52	723.51	1059.43	49.87
t3_d2_n8_r30.dat	0.61	332.7	663.91	49.54
t3_d2_n8_r40.dat	1.15	331.9	666.51	103.3
t3_d2_n8_r50.dat	0.91	323.44	664.06	101.03
t3_d2_n9_r10.dat	0.9	351.15	663.27	49.33
t3_d2_n9_r20.dat	0.95	355.21	662.55	99.94
t3_d2_n9_r30.dat	0.83	353.07	661.04	100.62
t3_d2_n9_r40.dat	1	353.74	910.42	478.67
t3_d2_n9_r50.dat	2.09	781.45	1543.8	695.26
t3_d2_n10_r10.dat	0.75	295.86	515.49	79.66
t3_d2_n10_r20.dat	0.61	301.44	521.04	183.49
t3_d2_n10_r30.dat	0.74	305.43	522.72	175.72
t3_d2_n10_r40.dat	0.87	295.48	522.37	391.21
t3_d2_n10_r50.dat	1.01	296.58	526.09	388.88

Tabla 21. Promedio de tiempos para la configuración t3_d3_n6-10_r10-50

FileName	SlimeMould	GreyWolf	DragonFly	QuantumEigensolver
t3_d3_n6_r10.dat	0.65	213.42	517.64	5.38
t3_d3_n6_r20.dat	0.58	215.64	519.46	7.7
t3_d3_n6_r30.dat	0.43	215.99	521.26	11.54
t3_d3_n6_r40.dat	0.54	215.53	518.57	20.7
t3_d3_n6_r50.dat	0.83	215.25	520.16	20.04
t3_d3_n7_r10.dat	0.61	238.18	524.06	11.61
t3_d3_n7_r20.dat	0.49	242.49	527.99	20.2
t3_d3_n7_r30.dat	0.59	237.64	514.71	19.06
t3_d3_n7_r40.dat	0.6	224.22	494.55	36.79
t3_d3_n7_r50.dat	0.6	228.7	487.19	37.11
t3_d3_n8_r10.dat	0.55	244.85	488.83	18.62
t3_d3_n8_r20.dat	0.66	246.06	490.99	35.99
t3_d3_n8_r30.dat	1.01	247.5	488.93	36.41
t3_d3_n8_r40.dat	0.53	246.09	491.13	74.48

t3_d3_n8_r50.dat	2.05	248.39	490.44	91.03
t3_d3_n9_r10.dat	0.99	348.92	603.77	44.28
t3_d3_n9_r20.dat	0.92	311.81	572.25	89.49
t3_d3_n9_r30.dat	0.84	317.08	566.79	87.7
t3_d3_n9_r40.dat	1.06	313.92	564.19	195.14
t3_d3_n9_r50.dat	2.66	300.54	539.3	170.88
t3_d3_n10_r10.dat	0.69	305.77	537.51	122.9
t3_d3_n10_r20.dat	1.21	433.56	779.25	262.38
t3_d3_n10_r30.dat	1.42	444.98	731.99	172.73
t3_d3_n10_r40.dat	0.85	285	519.74	391.3
t3_d3_n10_r50.dat	0.78	292.7	526.63	393.36

5.6 ANÁLISIS DE RESULTADOS

Con los resultados del promedio de fitness incluidos en las tablas de resultados para las ejecuciones de instancias no correlacionadas, débilmente correlacionadas y fuertemente correlacionadas, de manera general se puede afirmar que con los algoritmos SMA y QuantumEigensolver se logra obtener una solución óptima el 100% de las veces, lo cual concuerda con la tasa de éxito reportada en los archivos de solución incluidos en el **Anexo E**.

También se observa que los algoritmos QDGWO y DA+IAMDA presentan inconvenientes al tratar de encontrar soluciones a problemas que tienen un valor de r bajo (r=10), lo cual indica que estos algoritmos no logran ser eficientes cuando se tienen artículos con un peso relativamente bajo, lo cual permitiría llenar la mochila con la mayor cantidad de artículos disponibles.

En la medida que la complejidad del problema aumenta, el algoritmo QDGWO va incrementando su tasa de efectividad; Además, aunque en este mismo escenario el algoritmo DA+IAMDA disminuye la tasa de efectividad, los valores de decremento son mínimos, manteniendo su superioridad frente a QDGWO.

Respecto al tiempo de respuesta reportado se puede analizar que el algoritmo más eficiente (con un menor tiempo promedio de ejecución) es SMA, seguido de QuantumEigensolver. Aunque los dos algoritmos presentan un incremento en el tiempo de ejecución a medida que el problema aumenta de complejidad, QuantumEigensolver presenta un crecimiento exponencial mientras que SMA lo hace de manera cuadrática.

Con respecto a DA+IAMDA y QDGWO, se observa que QDGWO se toma un tiempo considerablemente mayor que el resto de los algoritmos, pero al igual que DA+IAMDA en general los dos mantienen un tiempo constante sin importar el aumento en la dificultad de los problemas.

5.6.1 Análisis estadístico de los datos aplicando un test de Friedman

Con los resultados obtenidos de la ejecución de las pruebas en el marco de trabajo propuesto (consignados en el **Anexo E**), se procedió a realizar un test de Friedman usando el software KEEL (www.keel.es).

Para determinar si la diferencia entre las medias es estadísticamente significativa, se comparó el valor p con el nivel de significancia específico para aceptar o rechazar la hipótesis nula. La hipótesis nula indica que las medias de población son similares. Por lo general, un nivel de significancia (denotado como α o alfa) de 0.05 funciona adecuadamente. Un nivel de significancia de 0.05 indica un riesgo de 5% de concluir que existe una diferencia cuando no hay una diferencia real, o en su defecto que se tiene un 95% de confianza en los resultados. A continuación, se presentan los resultados obtenidos.

5.6.1.1 Ranking de los algoritmos según el test de Friedman

Los resultados del del test de Friedman para las diferentes configuraciones de mochila son consignados en las **Tabla 22**, **Tabla 23** y **Tabla 24**. Bajo la hipótesis nula, se establece que todos los algoritmos son equivalentes, por lo que el rechazo de esta hipótesis implica la existencia de diferencias en el rendimiento de los algoritmos estudiados. Es preciso tener en cuenta que el ranking más bajo es el mejor y en estas tablas aparece en negrita, de la misma forma los valores p que con menores a 0.05 y que hacen el ranking estadísticamente significativo también están en negrita.

Tabla 22. Ranking promedio en instancias de dificultad fácil

Algoritmo	**Ranking**		
	No correlacionado	**Débilmente correlacionada**	**Fuertemente correlacionado**
SlimeMould	**2.1333**	**2.1**	**2.1167**
QuantumEigensolver	**2.1333**	**2.1**	**2.1167**
GreyWolf	2.3667	2.6	2.5833
DragonFly	3.3667	3.2	3.1833
Valor p (p<0.05)	**3.1837265678E-4**	**0.0020336671**	**0.0031606892**

Tabla 23. Ranking promedio en instancias de dificultad media

Algoritmo	**Clasificación**		
	No correlacionado	**Medianamente correlacionada**	**Fuertemente correlacionado**
SlimeMould	**2.1333**	**1.7167**	**2**
QuantumEigensolver	**2.1333**	**1.7167**	**2**
GreyWolf	2.75	3.7	3.2333
DragonFly	2.9833	2.8667	2.7667
Valor p (p<0.05)	**0.01717499646**	**1.0998069122E-10.**	**1.730133959E-4.**

Tabla 24. Ranking promedio en instancias de dificultad difícil

Algoritmo	Clasificación		
	No correlacionado	Medianamente correlacionada	Fuertemente correlacionada
SlimeMould	**1.82**	**2.08**	**1.82**
QuantumEigensolver	**1.82**	**2.08**	**1.82**
GreyWolf	3.02	2.8	3.24
DragonFly	3.34	3.04	3.12
Valor p (p<0.05)	**2.8356886880E-6**	**0.0116396714**	**3.9013402198E-6**

La prueba no paramétrica de Friedman establece bajo la hipótesis nula, que todos los rankings son válidos ya que se rechaza la hipótesis nula, es decir, se acepta que las medias son diferentes, y esto implica la existencia de diferencias entre el rendimiento de los algoritmos estudiados.

5.6.1.2 Comparaciones con el Post Hoc - Holm

Se usó también el procedimiento de comparación múltiple (Post Hoc) para revisar las diferencias de cada algoritmo frente a los otros en cada uno de los rankings. Este procedimiento funciona con un algoritmo de control (Holm), el cual es un procedimiento escalonado que prueba secuencialmente las hipótesis ordenadas según el grado de diferencia significativa. Si el valor p del ANOVA es menor que el nivel de significancia, se puede rechazar la hipótesis nula y concluir que tenemos evidencia suficiente para decir que al menos una de las medias de los grupos es diferente de las demás. A continuación, la **Tabla 25**, la **Tabla 26** y la **Tabla 27** muestran los resultados de estas comparaciones. Se resalta aquellas comparaciones que rechazan la hipótesis nula (medias iguales) por que tienen un valor p ajustado por Holm menor igual al definido al inicio de la columna.

Tabla 25. Post hoc para α = 0,05 en instancias de dificultad fácil

Algoritmo	No correlacionado p ≤ 0.016667		Medianamente correlacionada p ≤ 0.0125		Fuertemente correlacionada p ≤ 0.0125	
	p	Holm	p	Holm	p	Holm
SlimeMould vs. DragonFly	**0.000216**	0.008333	**0.000967**	0.008333	**0.001374**	0.008333
QuantumEigensolver vs DragonFly	**0.000216**	0.01	**0.000967**	0.01	**0.001374**	0.01
GreyWolf vs. DragonFly	**0.0027**	0.0125	0.071861	0.0125	0.071861	0.0125
SlimeMould vs. GreyWolf	0.483927	0.0125	0.133614	0.016667	0.161513	0.016667
QuantumEigensolver vs. GreyWolf	0.483927	0.025	0.133614	0.025	0.161513	0.025
SlimeMould vs. QuantumEigensolver	1	0.05	1	0.05	1	0.05

Con los resultados de la **Tabla 25**, para las instancias de dificultad fácil y sin importar el tipo de correlación se puede decir que tanto SlimeMould como QuantumEigensolver obtienen mejores resultados que DragonFly, es decir **{SlimeMould, QuantumEigensolver} > {DragonFly}**, donde el símbolo mayor significa que obtienen mejores resultados (símbolo de dominancia). No se puede establecer relación de dominancia entre los algoritmos SlimeMould, QuantumEigensolver y GreyWolf. Además, en las instancias no correlacionadas se puede decir que **{GreyWolf} > {DragonFly}.**

Tabla 26. Post hoc para α = 0,05 en instancias con dificultad media

Algoritmo	**No correlacionado** $p \leq 0.008333$		**Medianamente correlacionada** $p \leq 0.05$		**Fuertemente correlacionada** $p \leq 0.0125$	
	p	**Holm**	**p**	**Holm**	**p**	**Holm**
SlimeMould vs. Greywolf	0.010772	0.008333	**0**	0.008333	**0.000216**	0.008333
QuantumEigensolver vs GreyWolf	0.010772	0.01	**0**	0.01	**0.000216**	0.01
Slimemould vs. DragonFly	0.064314	0.0125	**0.000561**	0.0125	0.021448	0.0125
QuantumEigensolver vs. DragonFly	0.064314	0.016667	**0.000561**	0.016667	0.021448	0.016667
DragonFfly vs. GreyWolf	0.483927	0.025	**0.012419**	0.025	0.161513	0.025
SlimeMould vs. QuantumEigensolver	1	0.05	1	0.05	1	0.05

Con los resultados de la **Tabla 26**, para las instancias no correlacionadas se puede decir que no hay diferencias en los resultados obtenidos por los algoritmos, por lo que no se puede establecer relación de dominancia entre ellos.

En instancias medianamente correlacionadas se puede ver que se puede decir que **{SlimeMould, QuantumEigensolver} > {DragonFly} > {GreyWolf}**, pero no se puede establecer relación de dominancia entre SlimeMould y QuantumEigensolver.

En instancias fuertemente correlacionadas, se observa que tanto SlimeMould somo QuantumEigensolver obtienen mejores resultados que GreyWolf, es decir, **{SlimeMould, QuantumEigensolver} > {GreyWolf}**, pero no se puede establecer relación de dominancia entre SlimeMould, QuantumEigensolver y DragonFly.

Con los resultados de la **Tabla 27**, para las instancias con dificultad alta o difícil y que son no correlacionadas o que por el contrario son fuertemente correlacionadas se puede decir que **{SlimeMould, QuantumEigensolver} > {DragonFly, GreyWolf}**, pero no se puede relación de dominancia entre SlimeMould y QuantumEigensolver y tampoco entre DragonFly y GreyWolf.

Para las instancias difíciles y con mediana correlación sólo se puede decir que **{SlimeMould, QuantumEigensolver} > {DragonFly}.**

Tabla 27. Post hoc para α = 0,05 en instancias con dificultad difícil

Algoritmo	**No correlacionado** **p ≤ 0.025**		**Medianamente correlacionada** **p ≤ 0.008333**		**Fuertemente correlacionada** **p ≤ 0.025**	
	p	**Holm**	**p**	**Holm**	**p**	**Holm**
SlimeMould vs. DragonFly	**0.000031**	0.008333	**0.008562**	0.008333	**0.000371**	0.0125
QuantumEigensolver vs DragonFly	**0.000031**	0.01	**0.008562**	0.01	**0.000371**	0.016667
SlimeMould vs. GreyWolf	**0.001015**	0.0125	0.048632	0.0125	**0.000101**	0.008333
QuantumEigensolver vs GreyWolf	**0.001015**	0.016667	0.048632	0.016667	**0.000101**	0.01
GreyWolf vs. DragonFly	0.380836	0.025	0.511009	0.025	0.742433	0.025
SlimeMould vs. QuantumEigensolver	1	0.05	1	0.05	1	0.05

Esta página ha sido dejada intencionalmente en blanco.

CAPÍTULO 6

6 CONCLUSIONES Y TRABAJO FUTUROS

6.1 CONCLUSIONES

En este documento se ha detallado una estrategia para dar solución al problema de optimización de la mochila binaria (un problema NP-completo), haciendo uso de un algoritmo de Computación Cuántica Adiabática, detallando la manera como se mapea el problema a un modelo de Ising.

Para poder realizar pruebas con un conjunto de datos diverso (nivel de dificultad y nivel de correlación de los pesos y valores de los elementos a incluir o no en la mochila), se desarrolló un marco de trabajo que tiene plena capacidad de generar conjuntos de datos aleatorios que se configuran por tipo de correlación, dificultad de los problemas a generar, cantidad de ítems y el rango máximo de pesos que pueden ser fijados para cada artículo de mochila. La variedad en los valores del rango permite evaluar el desempeño de los algoritmos con mochilas que poseen elementos con un alto peso y por ello tienden a soluciones que no llegan a cubrir toda la capacidad disponible.

Los resultados obtenidos por parte del algoritmo cuántico se compararon con tres algoritmos basados en metaheurísticas poblacionales seleccionados del estado del arte teniendo como métricas de comparación el promedio de los mejores fitness y el promedio de los tiempos de ejecución obtenidos de 31 repeticiones por cada problema evaluado (la medía y el teorema del límite central).

El algoritmo cuántico logra encontrar la solución óptima a todos los problemas de la mochila con los que se evaluó e inclusive le gana en todos los aspectos a dos de los algoritmos comparados (GreyWolf y DrafgonFly), aunque el resultado no es estadísticamente significativo en todos los casos. Por otro lado, se observa que muy a pesar de que las instancias de evaluación son de baja dimensionalidad, se evidencia un crecimiento exponencial del tiempo de ejecución en la medida que aumenta la complejidad de los problemas para estos dos algoritmos. En el tiempo de ejecución el algoritmo que reporta los mejores (menores) valores es el SMA.

Los tres algoritmos seleccionados para realizar la comparación son metaheurísticas poblacionales, esto era de esperarse ya que en el estado del arte se había notado una preferencia por este tipo de algoritmos.

Para abordar problemas de mayor complejidad (mayor número de elementos) se hace necesario utilizar una computadora de mayor capacidad o idealmente un computador cuántico real con una gran cantidad de qubits. Para tomar el segundo camino, se hace necesario establecer alianzas con empresas como IBM o Google

quienes están desarrollando y probando los computadores cuánticos de mayor capacidad que existen hoy en día.

6.2 TRABAJO FUTUROS

Continuando con la línea de investigación de computación cuántica desarrollada sobre Qiskit, se podría evaluar la posibilidad de crear una clase personalizada que dé solución a las matrices de Pauli y se envié como parámetro a la clase MinimunEigenSolver(), esto con el fin de probar diferentes mecanismos de configuración.

Debido a que la evaluación se realizó con algoritmos poblaciones, se hace necesario evaluar la efectividad del algoritmo cuántico con diferentes tipos de metaheurísticas, además de buscar corroborar que el algoritmo entrega resultados óptimos en más tipos de instancias.

El marco de trabajo propuesto se puede adaptar para resolver otros problemas de optimización combinatoria, como problemas de subconjuntos de sumas o el problema del vendedor viajero.

CAPÍTULO 7

7 BIBLIOGRAFIA

[1] R. Karp, “Reducibility Among Combinatorial Problems,” in *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, 2010, pp. 219–241.

[2] M. W. Coffey, “Adiabatic quantum computing solution of the knapsack problem,” pp. 1–22, 2017.

[3] H. Wang, L. Ma, H. Zhang, and G. Li, “Quantum-inspired ant algorithm for knapsack problems,” *J. Syst. Eng. Electron.*, vol. 20, no. 5, pp. 1012–1016, 2009.

[4] S. Martello and P. Toth, “Algorithms for Knapsack Problems,” *North-holl. Math. Stud.*, vol. 132, no. C, pp. 213–257, 1987.

[5] F. Gurski, C. Rehs, and J. Rethmann, “Knapsack problems: A parameterized point of view,” *Theor. Comput. Sci.*, vol. 775, pp. 93–108, 2019.

[6] M. Assi and R. A. Haraty, “A Survey of the Knapsack Problem,” *ACIT 2018 - 19th Int. Arab Conf. Inf. Technol.*, pp. 1–6, 2019.

[7] H. M. Salkin and C. A. de Kluyver, “The knapsack problem: a survey*,” vol. 22, no. 1, pp. 127–144, 1975.

[8] D. Blado and A. Toriello, “Relaxation Analysis for the Dynamic Knapsack Problem with Stochastic Item Sizes,” *SIAM J. Optim.*, vol. 29, no. 1, pp. 1–30, Jan. 2019.

[9] D. Pisinger, “Where are the hard knapsack problems?,” *Comput. Oper. Res.*, vol. 32, no. 9, pp. 2271–2284, 2005.

[10] P. Vickram, A. S. Krishna, and V. S. Srinivas, “A Survey on Design Paradigms to solve 0/1 Knapsack Problem,” *Int. J. Sci. Eng. Res.*, vol. 7, no. 11, pp. 112–117, 2016.

[11] J. Li and W. Li, “A new quantum evolutionary algorithm in 0-1 knapsack problem,” *Commun. Comput. Inf. Sci.*, vol. 986, pp. 142–151, 2019.

[12] R. Wang, N. Guo, F. Xiang, and J. Mao, “An improved quantum genetic algorithm with mutation and its application to 0-1 knapsack problem,” *Intemational Conf. Meas. Inf. Control*, no. M Ic, pp. 484–488, 2012.

[13] S. Bolos, “GitHub - sorin-bolos/QiskitCampAsia2019,” 2019. [Online]. Available: https://github.com/sorin-bolos/QiskitCampAsia2019. [Accessed: 26-Mar-2020].

[14] M. Vogel, *Quantum Computation and Quantum Information, by M.A. Nielsen and I.L. Chuang*, vol. 52, no. 6. 2011.

[15] T. Albash and D. A. Lidar, “Adiabatic quantum computation,” *Rev. Mod. Phys.*, vol. 90, no. 1, p. 015002, 2018.

[16] C. A. Vega Fernández and J. S. Ramírez Celis, “Computación Cuántica: Implementación De Algoritmos De Shor Y Grover En El Computador Cuántico De Ibm,” Escuela colombiana de Ingenieria Julio Garavito, 2017.

[17] S. A. Hadfield, “Quantum Algorithms for Scientific Computing and Approximate Optimization,” pp. 1–264, 2018.

[18] D. López-Sandoval and C.-A. Cobos-Lozada, “Adiabatic Quantum Computing applied to the solution of the Binary Knapsack Problem,” *Rev. Ibérica Sist. e Tecnol. Informação*, vol. In Press, 2020.

[19] D. López Sandoval, “Quantum Algorithm to solve Binary Knapsack Problem,” 2022.

[Online]. Available: https://github.com/DaniloLopez/QuantumAlgorithmToSolveKnapsackProblem. [Accessed: 18-Mar-2022].

[20] T. Hey, “Quantum computing: an introduction,” *Comput. Control Eng.*, vol. 10, no. 3, pp. 105–112, 1999.

[21] A. Narayanan, “Quantum computing for beginners,” *Proc. 1999 Congr. Evol. Comput. CEC 1999*, vol. 3, pp. 2231–2238, 1999.

[22] P. W. Shor, “Algorithms for Quantum Computation: Discrete Logarithms and Factoring,” *Proc. 35th Annu. Symp. Found. Comput. Sci.*, vol. 59, no. 3, pp. 124–134, 1994.

[23] L. K. Grover, “A fast quantum mechanical algorithm for database search,” *Proc. 28th Annu. ACM Symp. Theory Comput.*, vol. 41, no. 3, pp. 212–221, 1996.

[24] Z. Laboudi and S. Chikhi, “Comparison of genetic algorithm and quantum genetic algorithm,” *Int. Arab J. Inf. Technol.*, vol. 9, no. 3, pp. 243–249, 2012.

[25] A. Layeb, “A novel quantum inspired cuckoo search for knapsack problems,” *Int. J. Bio-Inspired Comput.*, vol. 3, no. 5, pp. 297–305, 2011.

[26] Y. Cao, S. Jiang, D. Perouli, and S. Kais, “Solving Set Cover with Pairs Problem using Quantum Annealing,” *Sci. Rep.*, vol. 6, no. 1, pp. 1–15, 2016.

[27] A. Lucas, “Ising formulations of many NP problems,” *Front. Phys.*, vol. 2, pp. 1–14, 2014.

[28] G. Benenti, G. Casati, and G. Strini, *Principles of quantum computation and information: Volume II: Basic tools and special topics*, vol. I. 57 Shelton Street, Covent Garden, London WC2H 9HE: World Scientific Publishing Co. Pte. Ltd., 2007.

[29] S. G. Brush, “History of the Lenz-Ising model,” *Rev. Mod. Phys.*, vol. 39, no. 4, pp. 883–893, 1967.

[30] Z. Bian, F. Chudak, W. Macready, and G. Rose, “The Ising model: teaching an old problem new tricks,” *D-Wave Syst.*, pp. 1–32, 2010.

[31] Y. Zhou, X. Chen, and G. Zhou, “An improved monkey algorithm for a 0-1 knapsack problem,” *Appl. Soft Comput. J.*, vol. 38, pp. 817–830, 2016.

[32] W. Zouari, I. Alaya, and M. Tagina, “A hybrid ant colony algorithm with a local search for the strongly correlated knapsack problem,” *Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl. AICCSA*, vol. 2017-Octob, pp. 527–533, 2018.

[33] D. Sapra, R. Sharma, and A. P. Agarwal, “Comparative study of metaheuristic algorithms using Knapsack Problem,” *Proc. 7th Int. Conf. Conflu. 2017 Cloud Comput. Data Sci. Eng.*, pp. 134–137, 2017.

[34] P. H. Nguyen, D. Wang, and T. K. Truong, “A novel binary social spider algorithm for 0-1 knapsack problem,” *Int. J. Innov. Comput. Inf. Control*, vol. 13, no. 6, pp. 2039–2049, 2017.

[35] M. J. Islam, X. Li, and Y. Mei, “A time-varying transfer function for balancing the exploration and exploitation ability of a binary PSO,” *Appl. Soft Comput. J.*, vol. 59, pp. 182–196, 2017.

[36] M. El-Shafei, I. Ahmad, and M. G. Alfailakawi, “Hardware accelerator for solving 0–1 knapsack problems using binary harmony search,” *Int. J. Parallel, Emergent Distrib. Syst.*, vol. 33, no. 1, pp. 87–102, 2018.

[37] Y. Feng, J. Yang, C. Wu, M. Lu, and X. J. Zhao, “Solving 0–1 knapsack problems by chaotic monarch butterfly optimization algorithm with Gaussian mutation,” *Memetic Comput.*, vol. 10, no. 2, pp. 135–150, 2018.

[38] H. S. Alamri, K. Z. Zamli, M. F. Ab Razak, and A. Firdaus, “Solving 0/1 knapsack problem using opposition-based whale optimization algorithm (OWOA),” *ACM Int.*

Conf. Proceeding Ser., vol. Part F1479, pp. 135–139, 2019.
[39] F. B. Ozsoydan, “Effects of dominant wolves in grey wolf optimization algorithm,” *Appl. Soft Comput. J.*, vol. 83, p. 105658, 2019.
[40] B. Abdollahzadeh, S. Barshandeh, H. Javadi, and N. Epicoco, “An enhanced binary slime mould algorithm for solving the 0–1 knapsack problem,” *Eng. Comput.*, no. 0123456789, 2021.
[41] L. Wang, R. Shi, and J. Dong, “A hybridization of dragonfly algorithm optimization and angle modulation mechanism for 0-1 knapsack problems,” *Entropy*, vol. 23, no. 5, pp. 1–24, 2021.
[42] Y. Wang and W. Wang, “Quantum-inspired differential evolution with greywolf optimizer for 0-1 knapsack problem,” *Mathematics*, vol. 9, no. 11, 2021.
[43] D. Pisinger, “David Pisinger’s optimization codes.” [Online]. Available: http://hjemmesider.diku.dk/~pisinger/codes.html.
[44] D. Pisinger, “Core problems in knapsack algorithms,” *Oper. Res.*, vol. 47, no. 4, pp. 570–575, 1999.
[45] S. Martello, D. Pisinger, and P. Toth, “Dynamic programming and strong bounds for the 0-1 Knapsack Problem,” *Manage. Sci.*, vol. 45, no. 3, pp. 414–424, 1999.
[46] K. S. Pratt, “Design Patterns for Research Methods: Iterative Field Research,” *AAAI Spring Symp. Exp. Des. Real*, no. 1994, pp. 1–7, 2009.
[47] D. Greenberg and S. Wörner, “Three New Directions for Quantum Algorithms & Applications in Qiskit,” 2020. [Online]. Available: https://www.ibm.com/blogs/research/2020/05/new-qiskit-quantum-algorithms-applications/.