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

****

**DANILO LÓPEZ SANDOVAL**

**Director: Ph.D. 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 202**

TABLA DE CONTENIDO

[LISTA DE FIGURAS 4](#_Toc99434105)

[Capítulo 1 6](#_Toc99434106)

[1 Introducción 6](#_Toc99434107)

[1.1 Planteamiento del problema 6](#_Toc99434108)

[1.2 Aportes del proyecto 8](#_Toc99434109)

[1.3 Objetivos 9](#_Toc99434110)

[1.3.1 Objetivo general 9](#_Toc99434111)

[1.3.2 Objetivos específicos 9](#_Toc99434112)

[1.4 Resultados obtenidos 9](#_Toc99434113)

[1.5 Estructura de la monografía 10](#_Toc99434114)

[Capítulo 2 12](#_Toc99434115)

[2 Contexto teórico y estado del arte 12](#_Toc99434116)

[2.1 Contexto teórico 12](#_Toc99434118)

[2.1.1 Computación cuántica 12](#_Toc99434119)

[2.1.2 Computación cuántica adiabática (AQC) 13](#_Toc99434120)

[2.1.3 Hamiltoniano en computación cuántica adiabática 14](#_Toc99434121)

[2.1.4 Modelo Ising 14](#_Toc99434122)

[2.2 Estado del Arte 15](#_Toc99434123)

[2.2.1 Trabajos previos en el ámbito de algoritmos clásicos 15](#_Toc99434124)

[2.2.2 Trabajos previos en el ámbito de computación cuántica 19](#_Toc99434125)

[CAPÍTULO 3 21](#_Toc99434126)

[3 Marco de trabajo para realizar la evaluación y comparación de los algoritmos seleccionados 21](#_Toc99434128)

[3.1 Descripción del marco de trabajo 21](#_Toc99434131)

[3.1.1 Modulo principal (main) 23](#_Toc99434132)

[3.1.2 Modulo generador 25](#_Toc99434133)

[3.1.3 Módulo de archivos 27](#_Toc99434134)

[3.1.4 Módulo de algoritmos 29](#_Toc99434135)

[CAPÍTULO 4 32](#_Toc99434136)

[4 PROBLEMA DE LA MOCHILA BINARIA ABORDADO MEDIANTE COMPUTACION CUANTICA 32](#_Toc99434137)

[4.1 Metodología de desarrollo 32](#_Toc99434139)

[4.2 Como abordar un problema de optimización mediante el mapeo de una función de costo a un hamiltoniano cuántico 33](#_Toc99434140)

[4.2.1 Solución con (n + C) qubits 33](#_Toc99434141)

[4.2.2 Solución con (n + [log2 C] + 1) qubits 34](#_Toc99434142)

[4.2.3 Mapeo de una función de costo a su hamiltoniano cuántico 36](#_Toc99434143)

[4.3 Tres nuevas direcciones para aplicaciones y algoritmos cuánticos QA&A en Qiskit 37](#_Toc99434144)

[4.4 Descripción del algoritmo general y sus componentes para dar solución al problema de la mochila con Qiskit 38](#_Toc99434145)

[4.4.1 Librerías requeridas 38](#_Toc99434146)

[4.4.2 Mapeo a un problema de Ising 38](#_Toc99434147)

[4.4.3 Encontrar el estado base de un Hamiltoniano 39](#_Toc99434148)

[CAPÍTULO 5 40](#_Toc99434149)

[5 EXPERIMENTOS Y RESULTADOS 40](#_Toc99434150)

[5.1 Parámetros de configuración 40](#_Toc99434151)

[5.2 Conjunto de datos 40](#_Toc99434152)

[5.3 algoritmos comparados 41](#_Toc99434153)

[5.4 protocolo de experimentación 41](#_Toc99434154)

[CAPÍTULO 6 42](#_Toc99434155)

[6 CONSCLUSIONES Y TRABAJOS FUTUROS 42](#_Toc99434156)

[6.1 CONSLUSIONES 42](#_Toc99434159)

[6.2 TRABAJOS FUTUROS 42](#_Toc99434160)

[CAPÍTULO 7 43](#_Toc99434161)

[7 BIBLIOGRAFIA 43](#_Toc99434162)

LISTA DE FIGURAS

[**Figura 1.** Diagrama de clases de alto nivel 17](#_Toc99333949)

[**Figura 2.** Diagrama de clases detallado del módulo principal 18](#_Toc99333950)

[**Figura 3.** Diagrama de clases detallado del módulo generador 20](#_Toc99333951)

[**Figura 4.** Diagrama de clases detallado del módulo de archivos 23](#_Toc99333952)

[**Figura 5.** Diagrama de clases del módulo de algoritmos 25](#_Toc99333953)

lista de tablas

[**Tabla 1.** Parámetros de configuración de los tres algoritmos seleccionados para la evaluación 39](#_Toc99371840)

[**Tabla 2.** Parámetros de ejecución para la generación de los conjuntos de datos de prueba 40](#_Toc99371841)

**LISTA DE ANEXOS**

Anexo A.

Anexo B.

Anexo C.

Anexo D.

Anexo E.

Anexo F.

Resumen

asdf

Capítulo 1

# Introducción

## 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], control de presupuestos, 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 (Pi = Wi), 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 siguientes 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** ∈ Z+, y un conjunto de ***n*** elementos (artículos o ítems), cada uno con un beneficio **Pi** ∈ Ζ+ y un peso **Wi** ∈ Ζ+ donde i = 1, 2, ..., n, se debe seleccionar un subconjunto de ***m*** elementos () 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 es un valor binario {0,1} que indica si el elemento *i* debe ir o no en la mochila.

|  |  |
| --- | --- |
|  | (1) |

Existen diferentes maneras de clasificar un problema de mochila binaria, entre las más destacadas se tiene una definición basada es 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 cuánticos 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 presente documento ha realizado 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 documento se plantea 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 planeta 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 planeta el uso de instancias de baja dimensionalidad (n < 20), esto debido a que los emuladores tienen restricciones para poder procesar problemas con mayor número de dimensiones.

## 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.

## Objetivos

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

### 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.

### 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 QiskitTM 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.

## 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 realiza 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. La documentación del código se presenta en el **Anexo B**.
* **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 C** muestra el 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 **“nombre”** y que se encuentra en proceso de envío a una revista o evento nacional o internacional (Ver **Anexo D)**.

## 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 origino el trabajo, los aportes al problema, también los objetivos (general y específicos) definidos en el anteproyecto, un breve resumen de los resultados obtenidos, aportes 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: FRAMEWORK PARA SOPORTAR LA EXPERIMENTACIÓN:** 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: 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 5: 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 6: BIBLIOGRAFIA**: Este último capítulo contiene las referencias bibliográficas de sitios web, artículos y libros consultados para la realización del proyecto.

Capítulo 2

# Contexto teórico y estado del arte



## Contexto teórico

### 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)**.

|  |  |
| --- | --- |
|  | (2) |

Donde denota más de un vector en algún espacio vectorial y representa una superposición lineal de la partícula dados los vectores de estado cuántico individuales. representan los valores de bit clásicos 0 y 1 respectivamente. y son números complejos tales que , 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 o uno con una probabilidad . 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 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].

### 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].

### 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 (), cuyo estado fundamental codifica la solución a un problema de interés, y otro hamiltoniano (), cuyo estado fundamental es trivial. Entonces, si se prepara un sistema cuántico para estar en el estado fundamental , 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].

|  |  |
| --- | --- |
|  | (3) |

Donde s ∈ [0, 1], es un parámetro de tiempo considerado como una función dependiente del tiempo s(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 y no conmutan (Dos operadores no conmutan cuando se cumple que [A, B] = AB - BA 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.

### 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 , y que interactúan entre sí con base en un campo de interacción [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 si ∈ {-1, +1} representa el spin de la i-ésima partícula.

|  |  |
| --- | --- |
|  | (4) |

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

|  |  |
| --- | --- |
|  | (5) |

es una matriz de Pauli[[1]](#footnote-1) que actúa sobre el i-ésimospin en un espacio de Hilbert de N qubits donde 𝕀 es una matriz identidad de 2 x 2 y , ∈ ℝ son coeficientes [27][26]. es la fuerza del campo aplicado sobre el i-ésimo spin y actúa como el campo de interacción entre los spines vecinos i, j [30]. Un estado fundamental es una superposición de todos los estados posibles en la base propia de [27] donde es la puerta NOT sobre el i-ésimo qubit [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].

### 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.

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 de 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 evaluó con 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 al 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 respuestas.

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 se 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 y 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, se consideran 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.

### 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 + [ 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 [42] se propone un algoritmo de evolución diferencial que combina algunos de los principios 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 con problemas que se evalúan con 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.

CAPÍTULO 3



# Marco de trabajo propuesto



## Descripción del marco de trabajo

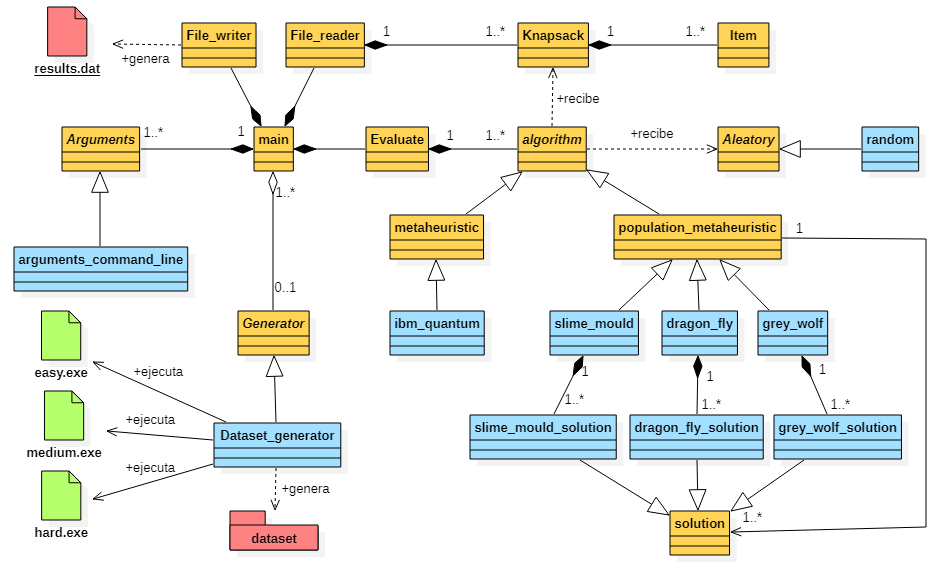
En la actualidad el manejo que se da en el ámbito del desarrollo de soluciones cuánticas en problemas de optimización se enfoca en modelos de circuitos cuánticos que trabajan con programación a bajo nivel, esto quiere decir con compuertas cuánticas que alteran el estado de un qubit el cual funciona sobre 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 tienen lenguajes de programación específicos 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 complejas y eficientes sobre una infraestructura de computación clásica.

Teniendo en cuenta que el modelo de circuitos de compuertas cuánticas definido para computación cuántica aún no está en la capacidad de dar solución a algoritmos complejos y que en el presente trabajo se desea utilizar algoritmos de computación cuántica que tuvieran la capacidad de resolver problemas de optimización, se decide escoger una librería desarrollada por el equipo de IBM para el estudio de soluciones cuánticas denominada Qiskit. Esta librería contiene un conjunto de herramientas y algoritmos desarrollados en base a la experimentación que se ha venido estudiando en una de las primeras computadoras cuánticas reales de IBM, por lo que tiene una base teórica robusta, cuenta con una documentación extensa y detallada, y además cuenta con todo el apoyo de la infraestructura que IBM ha venido creando en los últimos años.

En el proceso de revisión del estado del arte se puede vislumbrar un amplio abanico de algoritmos que se han venido utilizando para dar solución al problema de la mochila, donde una de las preocupaciones es tratar de encontrar un equilibrio entre exploración y explotación del espacio de búsqueda y lograr reducir el tiempo de respuesta utilizado como métrica de comparación de eficiencia y eficacia. Entre los algoritmos del estado del arte propuestos y la escogencia de los algoritmos que son objeto de evaluación de los mismos, se logra apreciar un uso recurrente de algoritmos que funcionan bajo metaheurísticas poblacionales y se plantea la necesidad de realizar estudios comparativos con instancias de prueba que tengan diversidad en cuanto a la correlación de elementos, dimensionalidad y tamaños de muestra. Esto nos da un camino de estudio y algunos criterios para la escogencia de los algoritmos que serán objeto de comparación en el presente trabajo.

En la **Figura 1** se muestra un diagrama de clases de alto nivel del algoritmo implementado. Las clases de color naranja corresponden al núcleo del marco de trabajo, y las clases de color azul son clases derivadas asociadas a implementaciones de algoritmos específico o implementaciones de clases abstractas, 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 hace 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.



**Figura 1.** Diagrama de clases de alto nivel

El marco de trabajo de divide en cuatro módulos:

* **Modulo principal (main):**

Modulo principal que controla el funcionamiento del algoritmo, definiendo la lógica requerida donde se permite crear y realizar pruebas sobre múltiples arquitecturas, algoritmos y conjuntos de datos.

* **Modulo generador:**

Se encarga de generar el conjunto de datos de acuerdo a los parámetros ingresados por línea de comandos.

* **Módulo de archivos:**

Se centra en realizar el cargue del conjunto de datos con los cuales se realiza la comparación de rendimiento de los algoritmos y se encarga de escribir los resultados en un documento de texto.

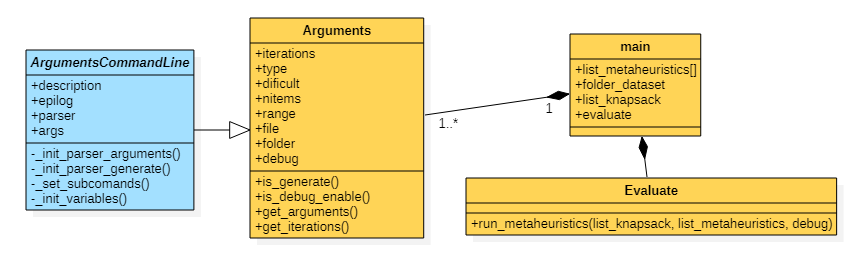
* **Módulo de algoritmos:**

Define la estructura base para lograr integrar nuevos esquemas de algoritmos.

En esta sección se explica brevemente cada uno de los módulos mencionados.

### Modulo principal (main)

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, tenemos las clases encargadas de la gestión de los argumentos ingresados por línea de comandos, y por el otro lado tenemos la clase principal y de evaluación encargadas de gestionar el funcionamiento del marco de trabajo.



**Figura 2.** Diagrama de clases detallado del módulo principal

#### Submódulo de argumentos

Se implementa una clase abstracta para que el aplicativo tenga la capacidad de ser mantenible a futuro y 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.

Debido a que el programa se ejecuta por 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.

Argumentos opcionales:

* **[-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 a modo de información para poder realizar una depuración.

* **[-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 contenga 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 profundiza en el apartado correspondiente al módulo generador.

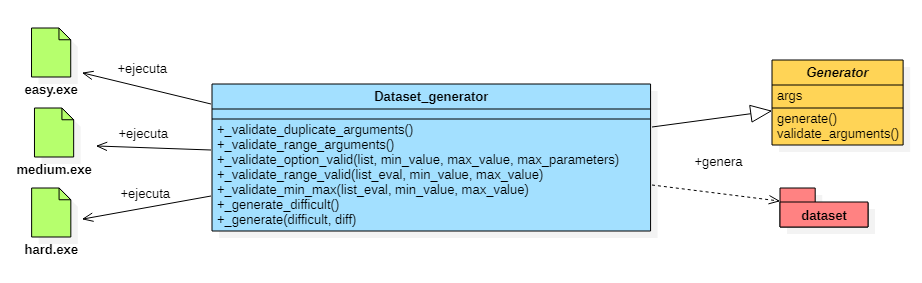
#### Submódulo principal (main)

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

* 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 a los parámetros ingresados por el usuario.
* Gestionar los algoritmos que serán objeto de pruebas.

### Modulo generador

En la **Figura 3** se muestra el módulo generador encargado de generar el conjunto de datos pertinentes de acuerdo a 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 permiten crear los conjuntos de datos necesarios.



**Figura 3.** Diagrama de clases detallado del módulo generador

#### 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 de mochila ampliamente reconocidos por la comunidad científica y 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**

Generador para construir instancias de prueba para el problema de la mochila binaria, este generador se describe en [44].

* **Medium.exe**

Generador avanzado para construir instancias de prueba para el problema de la mochila binaria, se consideran 14 tipos de instancias diferentes. este generador se describe en [45].

* **Hard.exe**

Generador para construir instancias de prueba complejas para el problema de la mochila binaria. este generador se describe en [9].

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

* **Tipo de correlación:**

Hace referencia al tipo de correlación que deben tener los elementos de la mochila.

* **Dificultad:**

Hace referencia a la dificultad que debe tener el conjunto de datos seleccionados, si se requiere una dificultad normal, el conjunto de datos se genera con el artefacto ***easy.exe***,si se requiere una dificultad mediael 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 que van a estar disponibles para almacenar en cada instancia de mochila.

* **Rango de los coeficientes:**

Hace referencia al valor máximo permitido para los valores de peso de cada elemento disponible para almacenar mochila.

Teniendo en cuenta las variables definidas anteriormente, en primera medida los archivos serán almacenados en una carpeta general la cual se nombra 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 al 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 siguiente estructura:

n C

p[0] w[0]

p[1] w[1]

:

p[n-1] w[n-1]

Donde **n** indica la cantidad de elementos, **C** indica el 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.

#### Generación del conjunto de datos

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

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} ].

### Módulo de archivos

En la **Figura 4** 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 archivos y otra encargada de la escritura de resultados en un archivo de salida.

#### FileReader

Esta clase se encarga de realizar la lectura de un archivo que contiene la información de una mochila. 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 una solución objetivo, esta será calculada por medio de un algoritmo de fuerza bruta que evalúa todas las posibles combinaciones y entre ellas selecciona la mejor. posteriormente se escribe el valor objetivo en el archivo seguido de la solución óptima. Al finalizar la lectura del archivo, en él se organizan los datos de la siguiente manera:

n C

p[0] w[0]

p[1] w[1]

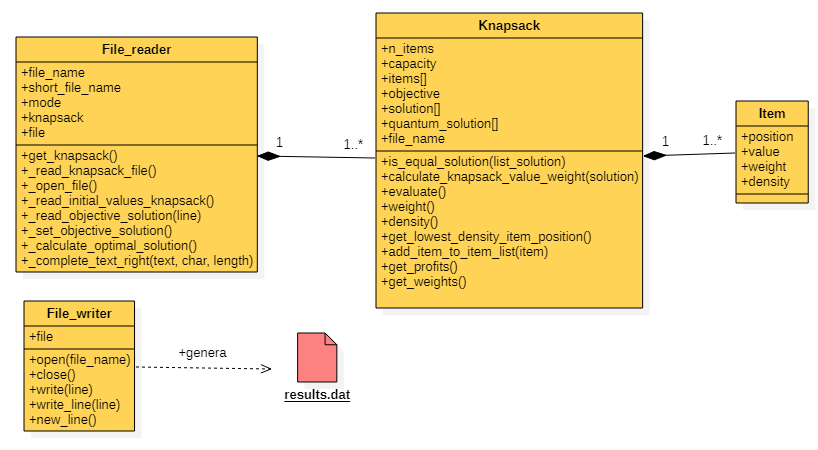
:

p[n-1] w[n-1]

obj

0 0 0 … n-1

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, **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 ceros o unos donde uno significa que el elemento va en la mochila y cero en caso contrario.



**Figura 4.** Diagrama de clases detallado del módulo de archivos

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()**.

#### FileWriter

Esta clase se encarga de realizar la escritura de los resultados de la ejecución de todas las iteraciones de los algoritmos seleccionados en dos archivos de texto para su posterior evaluación. Estos archivos son almacenados 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 el 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 Items
  + Capacidad de la mochila
  + Valor objectivo
  + Solucion

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 por cada algoritmo:

* + Tasa de exito
  + Promedio de fitness
  + Desviacion estandar
  + Valor del major fitness
  + Valor del peor fitness
  + Tiempo promedio
* **result\_[datetime]\_fitness.csv**,

Al inicio de este archivo se registra un encabezado que contiene el nombre del conjunto de datos evaluado seguido de los nombres de los algoritmos seleccionados para realizar el proceso de evaluación. En la primera ejecución, en una sola línea se registra el nombre del archivo que contiene 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.

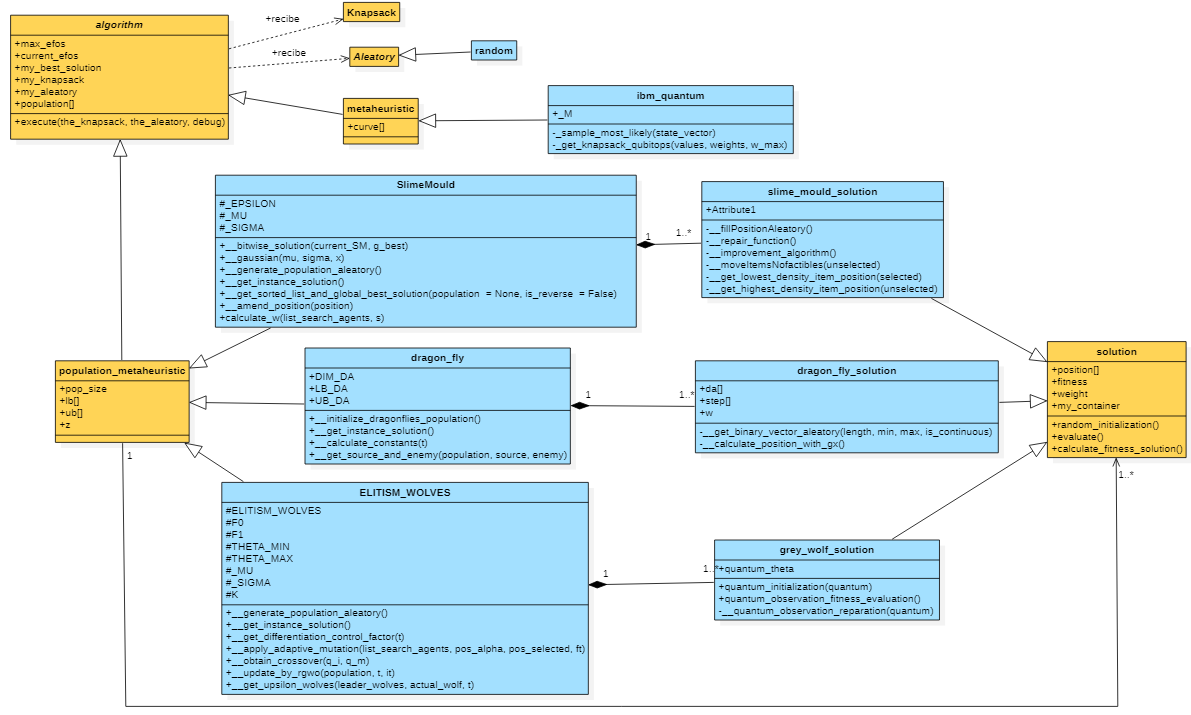
### Módulo de algoritmos

En la **Figura 5** 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 garantiza que se puedan adicionar nuevas metaheurísticas sin que se afecte el funcionamiento del marco de trabajo.

#### 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 que almacena la información de la mochila, 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 5.** Diagrama de clases del módulo de algoritmos

En la definición del método abstracto *execute(the\_knapsack, the\_aletory, 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 una instancia de mochila deseada y una estrategia de aleatoriedad. Se concibe desarrollarlo de esta manera para que el marco de trabajo tenga la capacidad de adaptarse a diferentes lógicas de ejecución permitiendo un alto nivel de mantenibilidad.

#### 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 algunos 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.

#### 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.

#### 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.

#### 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 operadoressirven de insumo para la clase **ExactEigensolver** la cual es la instancia de Qiskit Aqua encargada de encontrar la solución al problema de mochila.

#### IbmQuantumEigensolver

En esta clase se sobrescribe el método *execute(the\_knapsack, the\_aletory, debug)* heredado de la clase padre **Metaheuristic**, en ella se define el algoritmo que da solución al problema de mochila especifico 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.

CAPÍTULO 4

# PROBLEMA DE LA MOCHILA BINARIA ABORDADO MEDIANTE COMPUTACION CUANTICA



## Metodología de desarrollo

En este trabajo se utilizó el patrón de investigación iterativo (PII) propuesto por Pratt [46], para el desarrollo de los algoritmos propuestos en este capítulo. 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 una 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 nos da un acercamiento acerca de los problemas que puede afrontar cada algoritmo en un entorno real.

Posteriormente, 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 decide tomar como insumo tres generadores automáticos de instancias de mochila los cuales permiten construir los datos necesarios de acuerdo a 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 pretende 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 procede a implementar el módulo de archivos que permite realizar la gestión de los archivos generados y finalmente se inicia la construcción del módulo de algoritmos teniendo presente que se desea un sistema mantenible 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 ejecutables faltantes que realizan la generación del conjunto de datos, se adicionan un componente para realizar la gestión de los argumentos introducidos por línea de comandos y se implementan los algoritmos metaheurísticos que serán objeto de estudio. En paralelo con los ciclos de PII se realizan 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 realizarán actividades de documentación, escritura de la monografía y sus anexos; Para finalizar, en el ultimo ciclo de este proceso se inició la escritura de un artículo para divulgación científica.

Durante la última etapa se finiquitaron detalles relacionados con los pseudocódigos de los algoritmos y se realizó un proceso retrospectivo para identificar todos los aspectos necesarios para lograr una adecuada reproducibilidad de los resultados de las pruebas finales presentadas en el Capítulo 5.

## Como abordar un problema de optimización mediante el 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 los problemas resueltos 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 N.

Los dos modelos de solución se detallan a continuación ya que podrían servir como base para lograr resolver problemas de optimización similares.

### Solución con (n + C) qubits

Teniendo en cuenta la definición del problema de la mochila propuesta en la **Ecuación (1)** y trabajando con la versión binaria de la misma, se tienen *n* variables 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 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 y la capacidad máxima de la mochila: (*n* + C) → [].

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

|  |  |
| --- | --- |
|  | (6) |

Con 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 , el resto de para los otros valores de k. Si la capacidad de la mochila (C) fuese de 20, esta solución es factible y en ese caso sería igual a , eliminando así a . Pero si C tuviera u valor igual a 8, esta solución no sería factible y en este caso sería igual a haciendo que el valor de sea grande (positivo, alejando la solución del mínimo), y entre más se supere el peso máximo de la mochila, mayor será el valor de . Finalmente se tiene en la **Ecuación (7)**.

|  |  |
| --- | --- |
|  | (7) |

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

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

Realizando una modificación a la solución planteada anteriormente, la cual requiere (*n* + C) qubits, se desea reducir drásticamente la cantidad de giros adicionales que deben agregarse. Así, codificando la magnitud del peso máximo (C) como un valor binario, se define una variable entera M tal que , con M = ⌊⌋; De esta manera solo se necesitan (M+1) variables binarias: ... ], en lugar de C variables binarias: [ ... ], 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*+⌊⌋+1) qubits [27]. En la **Ecuación (8)** se presenta construido de acuerdo a la nueva representación de [2].

|  |  |
| --- | --- |
|  | (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 = [+1=5, por lo tanto, se tiene el vector = 10, que corresponde a la representación de 10 en binario de izquierda a derecha; Así, se cumple la condición 16 32. Esta solución es factible, por lo que , quedando eliminado . Por otro lado, si *C* = 9, se tendría el vector con , 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 = + esta dado por la **Ecuación (9)**, que utiliza una representación en base cero (o) para los vectores.

|  |  |
| --- | --- |
|  | (9) |

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

* La solución se representa como un vector = [ ... ] ∈ {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 = [ ... ] ∈ {0, 1}.
* El valor del peso de los elementos se representa como un vector W= [].
* El beneficio de los elementos se representa como un vector P = [].
* 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() se debe tener en cuenta que se desea maximizar la **Ecuación (1)** 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 = [… ], 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 y , con = [… … ]. 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 y se tenga un S = . El valor de L debe dominar la suma de los valores de P, esto es, . Si por el contrario , 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].

|  |  |
| --- | --- |
|  | (10) |
|  |

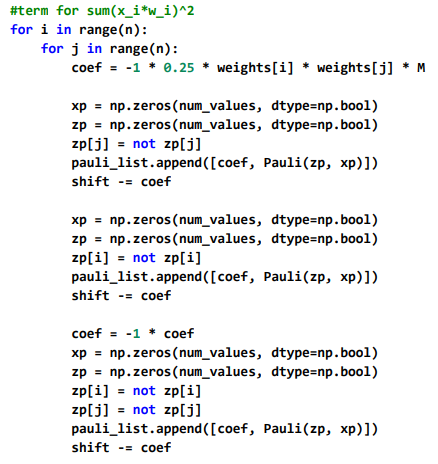
### Mapeo de una función de costo a su hamiltoniano cuántico

Al realizar el mapeo de las variables por (𝕀 - )/2, por (𝕀 - )/2 y eliminar el primer término que es constante, se obtiene la Ecuación (11), donde 𝕀 es una matriz identidad de orden *n+m* y 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: , se muestra su algoritmo en la **Figura 6**.

|  |  |
| --- | --- |
|  | (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.



**Figura 6.** Implementación del término para calcular los operadores de Pauli

## Tres nuevas direcciones para aplicaciones y algoritmos cuánticos QA&A en Qiskit

Durante la fase de ejecución de este trabajo nos topamos con 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 algorítmicos extremadamente sólidos, incluso uno sin algoritmos de caja negra en absoluto, puede ser más convincente”[47]. Es por ello 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 nuestra investigación, queda como base de estudio para entender el funcionamiento interno del módulo de optimización, ya que se continua 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.**

## Descripción del algoritmo general y sus componentes para dar solución al problema de la mochila con Qiskit

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 **Ecuacion (1)**.

Luego de obtener la **Ecuacion (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.

### 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 siguientes componentes:

* from *qiskit\_optimization.algorithms* import *MinimumEigenOptimizer*
* from *qiskit* import *Aer*
* from *qiskit.utils* import *algorithm\_globals, QuantumInstance*
* from *qiskit.algorithms* import *QAOA, NumPyMinimumEigensolver*
* from *qiskit\_optimization.applications* import *Knapsack*

### Mapeo a un problema de Ising

Para mapear el problema de mochila 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, de la siguiente forma*:*

* prob = Knapsack(

values=[x0, x1, … , xn-1],

weights=[w0, w1, … , wn-1],

max\_weight=C

)

* qp = prob.to\_quadratic\_program()

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

### 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 4.4.2, 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.

* meo = MinimumEigenOptimizer(

min\_eigen\_solver=NumPyMinimumEigensolver()

)

* result = meo.solve(qp)
* solution = result.x.astype(int).tolist()

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

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, 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

# 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 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.

## 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 de los tres algoritmos seleccionados para la evaluación

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

## 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 de las diferentes configuraciones para cada conjunto de datos.

**Tabla 2.** Parámetros de ejecución para la generación de los conjuntos de datos de prueba

|  |  |
| --- | --- |
| Conjunto de datos | Parámetros de ejecución |
| 1 | -i 31 generate -t 1 -d 1 -n 5 10 -r 10 50 |
| 2 | -i 31 generate -t 1 -d 2 -n 5 10 -r 10 50 |
| 3 | -i 31 generate -t 1 -d 3 -n 5 10 -r 10 50 |
| 4 | -i 31 generate -t 2 -d 1 -n 5 10 -r 10 50 |
| 5 | -i 31 generate -t 2 -d 2 -n 5 10 -r 10 50 |
| 6 | -i 31 generate -t 2 -d 3 -n 5 10 -r 10 50 |
| 7 | -i 31 generate -t 3 -d 1 -n 5 10 -r 10 50 |
| 8 | -i 31 generate -t 3 -d 2 -n 5 10 -r 10 50 |
| 9 | -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 obtenidos, se utilizaron para evaluar el desempeño de los algoritmos propuestos

## algoritmos comparados

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

* Slime Mould Algorithm – SMA
* Quantum-Inspired Differential Evolution with Grey Wolf Optimizer - QDGWO
* Dragonfly Algorithm Optimization and Angle Modulation Mechanism - DA+IAMDA.

Con esta selección se cubren algoritmos que incorporan mecanismos de solución novedosos o incorporan algún componente de computación cuántica. Los parámetros de configuración específicos de cada algoritmo se presentan en la **Tabla 3.**

**Tabla 3.** Parámetros de configuración específicos para cada algoritmo seleccionado

|  |  |
| --- | --- |
| 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 |

## protocolo de experimentación

La fase de experimentación se realizó en nueve etapas de ejecución. Cada etapa inicia con la puesta en marcha del algoritmo que se detalla en el **Capítulo 3** tomando los parámetros de ejecución indicados en la **Tabla 3**. Estos algoritmos fueron implementados sobre el marco de prueba propuesto tomando la lógica de programación que cada autor detalla en los pseudocodigos de los artículos seleccionados del estado del arte y se realizaron las adaptaciones pertinentes a cada uno para que se adecuara al marco de trabajo propuesto en cuestión.

Al momento de la ejecución, el algoritmo genera todo el conjunto de datos y los almacena en el fichero **files** del directorio del proyecto que se encuentra en el anexo 3, 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 las soluciones probables al conjunto de datos.

Para realizar la evaluación y las comparaciones entre los diferentes escenarios de evaluación obtenidos en base a la diversidad del conjunto de datos generado, se calcula la tasa de éxito, se realizaron mediciones promedio del máximo fitness encontrado y se calcula el tiempo promedio 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 el apartado de dedicado al numeral **3.1.3.2 FileWriter.**

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** para estudiar el comportamiento de cada algoritmo.

## 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 en las 31 ejecuciones por cada archivo.

### Resultados promedio de los mejores fitness encontrados

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 de la mochila.

#### Resultados de Fase 1 2 y 3 para instancias no correlacionadas

En las tablas 4, 5 y 6 se encuentra la información correspondiente a la ejecución de las pruebas para instancias no correlacionadas (parámetro t = 1).

**Tabla 4.** Resultados de promedio de fitness encontrados con la configuración t1\_d1\_n5-10\_r10-50

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FileName | Slime\_mould | Grey\_wolf | Dragon\_fly | 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. Resultados de promedio de fitness encontrados con la configuración t1\_d2\_n5-10\_r10-50

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FileName | Slime\_mould | Grey\_wolf | Dragon\_fly | 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. Resultados de promedio de fitness encontrados con la configuración t1\_d3\_n5-10\_r10-50

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FileName | Slime\_mould | Grey\_wolf | Dragon\_fly | 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 |

En estas tablas de resultados, de manera general se evidencia que el algoritmo DA obtuvo la mayor variabilidad de fitness en las tres fases evaluando instancias no correlacionadas, lo que nos indica una debilidad del algoritmo evaluando este tipo de instancias.

El algoritmo cuántico obtiene la solución adecuada en todas las iteraciones.

En relación a los tiempos de ejecución, el algoritmo que converge rápidamente hacia una solución viable es SMA, seguido del algoritmo cuántico y el algoritmo más demorado fue QDGWO.

#### Resultados de Fase 1 2 y 3 para instancias débilmente correlacionadas

En las tablas 4, 5 y 6 se encuentra la información correspondiente a la ejecución de las pruebas para instancias no correlacionadas (parámetro t = 2).

Tabla 7. Resultados de promedio de fitness encontrados con la configuración t2\_d1\_n5-10\_r10-50

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FileName | Slime\_mould | Grey\_wolf | Dragon\_fly | 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 8. Resultados de promedio de fitness encontrados con la configuración t2\_d2\_n5-10\_r10-50

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FileName | Slime\_mould | Grey\_wolf | Dragon\_fly | 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 9. Resultados de promedio de fitness encontrados con la configuración t2\_d3\_n5-10\_r10-50

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FileName | Slime\_mould | Grey\_wolf | Dragon\_fly | 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 |

En estas tablas de resultados, de manera general se evidencia que el algoritmo DA obtuvo la mayor variabilidad de fitness en las tres fases evaluando instancias no correlacionadas, lo que nos indica una debilidad del algoritmo evaluando este tipo de instancias.

Es claro que el algoritmo cuántico obtiene la solución adecuada en todas las iteraciones.

Analizando los resultados obtenidos para las mochilas que contienen 10 artículos , se logra apreciar que los algoritmos QDGWO y DA+IAMDA comienzan a presentar una leve variación negativa en los resultados a medida que aumenta el número de artículos y la correlación, evidenciando así una brecha en la optimización de mochilas con un número elevado de artículos.

En relación a los tiempos de ejecución, el algoritmo que converge rápidamente hacia una solución viable es SMA, seguido del algoritmo cuántico y el algoritmo más demorado fue QDGWO.

#### Resultados de Fase 1 2 y 3 para instancias fuertemente correlacionadas

En las tablas 4, 5 y 6 se encuentra la información correspondiente a la ejecución de las pruebas para instancias no correlacionadas (parámetro t = 3).

CAPÍTULO 6

# CONSCLUSIONES Y TRABAJOS FUTUROS



## CONSLUSIONES

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.

La variedad en los valores del rango nos 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 cubrir toda la capacidad disponible.

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 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.

## TRABAJOS FUTUROS

Teniendo en cuenta que el algoritmo cuántico mostró una debilidad concreta en la solución de un problema donde el óptimo se encuentra cuando la mochila queda desocupada en un 25% de su capacidad, se hace necesario que el Grupo se centre en mejorar la formulación de la función de costo y ampliar los tipos de problemas (instancias) que se usan en la evaluación para poder tener un panorama completo del rendimiento real del algoritmo.

CAPÍTULO 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/.

1. Matriz de 2x2 cuyo primo tiene vectores propios con valores propios 0,1 [27] [↑](#footnote-ref-1)